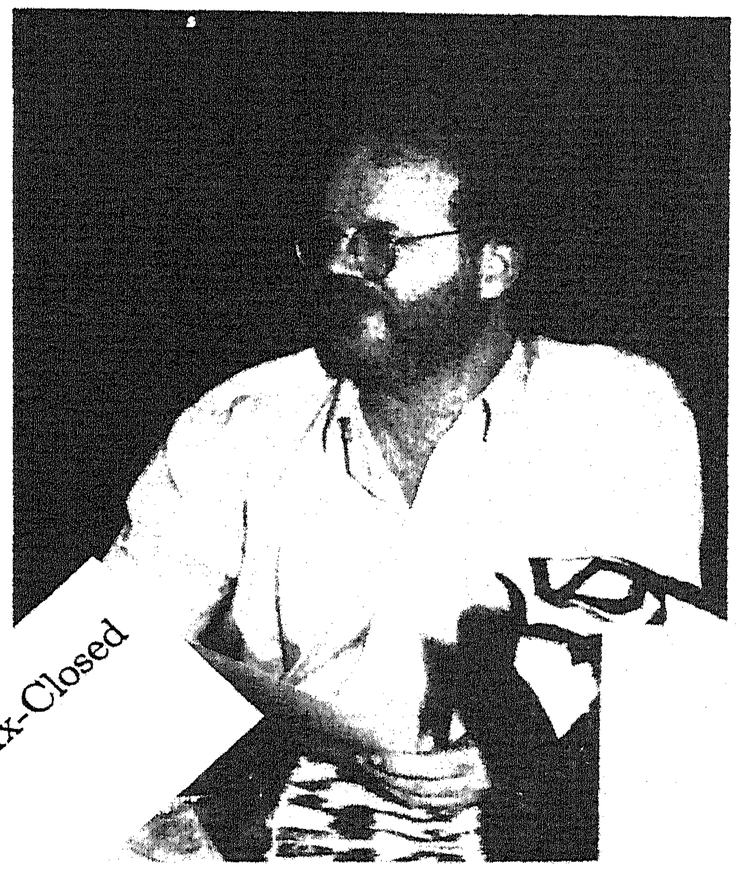
Founded 1989
Editor–in–chief:
Johan T. Jeuring

The Squiggolist

The Squiggolist, Volume 1, Number 2, November 1989

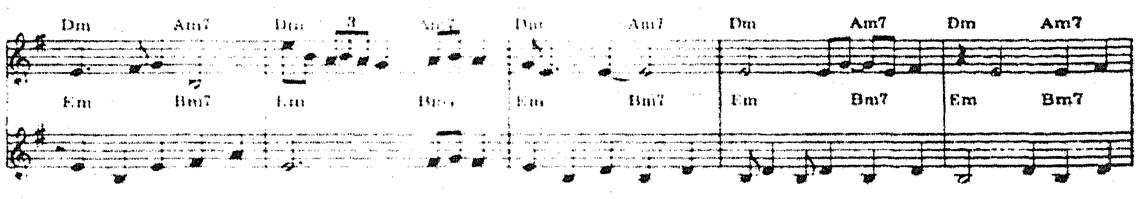
DEN HAAG — Old book discovered. In the vaults of the ministry of justice of the Netherlands, an old book has been discovered. The book contains the play 'Rule for rule' from Shakespeare, and is dated 's-Gravenhage 1690. Its main interest is its form: the book consists of thirty pages each of which consists of forty slips of paper. Apparently, the book has been produced by first printing it on a long slip of paper and then tearing and glueing this slip into the form of a book. Dutch historians claim that long before Turing wrote his 'On computable numbers, with an application to the Entscheidungsproblem', Christiaan Huygens had invented and built a 'Turing-Machine'. The fact that the ministry of [illegible] is built on the place where the Huygens family lived adds to the credibility of the claim.

DELFT — four death[illegible] killed [illegible] yesterday [illegible] to [illegible], the booby-[illegible]. While they [illegible] the trap, the [illegible] the four programmers the virus were [illegible] half of the building in which they were working was destroyed. Afterwards, a computer-security expert warned never to use booby-traps as protection to infection by viruses.



The Unique Derivative of a Prefix-Closed Predicate

Jeremy Gibbons



Guess the title

(advertisement)

Fight induction
Use UEP!

LOS ANGELOS — 'Avalanche' most popular language. Thirty percent of the companies in the USA use 'Avalanche' as their main computer programming language. Fortran has lost the first position it occupied last year to this relatively new language, the author of which is unknown. Since the author is unknown, no rights have to be paid for this piece of software, which is one of the reasons for its popularity. Another advantage of 'Avalanche' over other languages is the fact that it is an extremely well-designed and user-friendly language. According to an expert, the author looses billions of dollars on a program which it took him or her at least five years to write.

Whither application?

Carroll Morgan

LONDON — ODC
The [illegible]

WELLINGTON [illegible]

An Interesting Characterisation of Prefix-Closure

Jeremy Gibbons

[illegible] Europe in 1986 and has never returned since. He said that all his knowledge from notation he had learned from reading 'Algorithmics'.

[illegible] companies express serious doubts about the proper distribution of operators in future. 'I hope we can abide the distribution of operators,' a spokesman said, 'but if it does not work, we will mobilize our relations.'

For those who love calculating:

$$\curlyvee_{\oplus} = ((+\!\!+/)\circ) \circ (\tilde{*}) \circ (\circ(\tilde{\otimes})) \circ (*)$$

and

$$((*)\circ) \circ (\tilde{*}) \quad = \quad ((*)\circ) \circ (\circ) \circ (\circ)$$

Maarten M. Fokkinga

The least–effort cabinet formation

Johan Jeuring
Lambert Meertens

address: 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 second issue 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

This edition is sent to all attendants of the International Summer School on Constructive Algorithmics. Although it is not possible to subscribe to the Squiggolist, I might send the following editions provided a letter stating that receiving the Squiggolist is appreciated is sent to me. Furthermore, all new squiggol-experts are invited to submit their interesting ideas about squiggol.

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

Whither application?

Carroll Morgan

Abstract

Use of a right-assocative weakest-binding function-application operator can reduce the need for parentheses.

Function application, whether Eindhoven . or white space, usually associates to the left so that (using white space)

$$f\ g\ h\ x = ((f\ g)\ h)\ x.$$

Many derivations, however, involve expressions $f \cdot g \cdot h$, where the central dot $\cdot$ is functional composition. Since $\cdot$ is associative, there's no need for parentheses there; but when looking for a left-reduction, say, one writes

$$(f \cdot g \cdot h)\,(x{+\!\!+}[a])$$

to ensure the composition $f \cdot g \cdot h$ binds tighter than the application to $x{+\!\!+}[a]$. Subsequent calculation to distribute the $+\!\!+$ leftwards then requires

$$f(g(h(x{+\!\!+}[a]))).$$

And that is a lot of parentheses.

In denotational semantics — the Squiggol of the seventies — a similar problem arose. When using continuations, which are functions that represent 'the rest of the program', one would write

$$\mathcal{C}[\![F; G; H]\!]\rho\theta = \mathcal{C}[\![F]\!]\rho\,(\mathcal{C}[\![G]\!]\rho\,(\mathcal{C}[\![H]\!]\rho\,\theta))), \tag{1}$$

where ; denotes sequential composition of the imperative program fragments F, G, and H.

Mike Gordon[1] removed the parentheses by defining a functional composition operator ; that associates to the right and has weakest binding. Equation (1) above becomes

$$\mathcal{C}[\![F; G; H]\!]\rho\theta = \mathcal{C}[\![F]\!]\rho;\ \mathcal{C}[\![G]\!]\rho;\ \mathcal{C}[\![H]\!]\rho\,\theta,$$

with a nice pun between the syntactic semicolon (inside $[\![\cdots]\!]$) and the new semantic one.

A similar pun rewards the use of ; in Squiggol. Having already written $f \cdot g \cdot h$, then belatedly realising a pointwise argument is required, one need only attach a comma to each composition, from below:

[1] Gordon, M.J.C. *The denotational description of programming languages* Springer-Verlag 1979

$f \cdot g \cdot h$ becomes $f;g;h$,

↑ ↑

, ,

and now the argument $(x{+\!\!+}[a])$ can be added with no further parentheses at all. In fact, parentheses can be avoided altogether by writing

$$f;g;h;x{+\!\!+}[a].$$

Other equivalent expressions are (as *lgltm* points out)

$f \cdot g;h;x{+\!\!+}[a]$ and $f \cdot g \cdot h;x{+\!\!+}[a]$.

The Unique Derivative of a Prefix-Closed Predicate

Jeremy Gibbons
Oxford University

When we are considering a prefix-closed predicate p, we often talk about 'the derivative' of p; that is, the function δ such that

$$p\ (x +\!\!+ [a]) = p\ x \wedge \delta\ x\ a \tag{1}$$

Unfortunately, this doesn't make a great deal of sense. Both occurrences of the word 'the' in that first sentence are sloppy, since there may be many such derivatives. Informally, these derivatives may differ on what they assert about $p\ x$; since they are specified in the above context, the idempotence of $\wedge$ 'masks' all these assertions.

The question then naturally arises, "is there a 'canonical' derivative?" To which the answer is, yes. In fact, there are at least two intuitive and elegant canonical derivatives, the 'weakest' and the 'strongest'. The weakest derivative asserts nothing about $p\ x$; the strongest derivative asserts that $p\ x$ is true.

Thus, for any predicate p with (a) derivative δ, the functions w and s, given by

$$\begin{aligned} w\ x\ a &= p\ x \Rightarrow \delta\ x\ a \\ s\ x\ a &= p\ x \wedge \delta\ x\ a \end{aligned}$$

are also derivatives, and they satisfy the 'ordering'

$$s\ x\ a \;\Rightarrow\; \delta\ x\ a \;\Rightarrow\; w\ x\ a$$

As it happens, the strongest derivative is not particularly useful, since it is subsumed by p itself — $s\ x\ a = p\ (x +\!\!+ [a])$. However, I quite like the idea of the weakest derivative.

An Interesting Characterisation of Prefix-Closure

Jeremy Gibbons
Oxford University

Theorem 1 *A predicate p is prefix-closed iff it can be written in the form* all $q \cdot$ inits, *for some q which is satisfied by* $[\,]$.

Proof. Assume p is prefix-closed, i.e.,

$$\begin{aligned} p\ [\,] &= \mathsf{True} \\ p\ (x{+\!\!+}[a]) &= p\ x \wedge \delta\ x\ a \end{aligned}$$

for some δ (which is called the *derivative* of p). Let q be given by

$$\begin{aligned} q\ [\,] &= \mathsf{True} \\ q\ (x{+\!\!+}[a]) &= \delta\ x\ a \end{aligned}$$

Then

$$\begin{array}{ll} & (\mathsf{all}\ q \cdot \mathsf{inits})\ [\,] \\ = & \text{definition of inits} \\ & \mathsf{all}\ q\ [[\,]] \\ = & \text{definition of all} \\ & q\ [\,] \\ = & \text{definition of } q \\ & \mathsf{True} \\ = & p \text{ is prefix-closed} \\ & p\ [\,] \end{array}$$

and, assuming that (all $q \cdot$ inits) $x = p\ x$,

$$
\begin{array}{ll}
 & (\mathsf{all}\ q \cdot \mathsf{inits})\ (x{+\!\!+}[a]) \\
= & \text{definition of inits} \\
 & \mathsf{all}\ q\ (\mathsf{inits}\ x{+\!\!+}[x{+\!\!+}[a]]) \\
= & \mathsf{all}\ q\ (x{+\!\!+}y) = \mathsf{all}\ q\ x \wedge \mathsf{all}\ q\ y \\
 & \mathsf{all}\ q\ (\mathsf{inits}\ x) \wedge \mathsf{all}\ q\ [x{+\!\!+}[a]] \\
= & \text{definition of all} \\
 & \mathsf{all}\ q\ (\mathsf{inits}\ x) \wedge q\ (x{+\!\!+}[a]) \\
= & \text{definition of } q \\
 & \mathsf{all}\ q\ (\mathsf{inits}\ x) \wedge \delta\ x\ a \\
= & \text{inductive hypothesis} \\
 & p\ x \wedge \delta\ x\ a \\
= & p \text{ is prefix-closed} \\
 & p\ (x{+\!\!+}[a])
\end{array}
$$

This proves the implication from left to right. Now, assume conversely that $p = \mathsf{all}\ q \cdot \mathsf{inits}$, for some q which holds for $[\,]$. Then

$$
\begin{array}{ll}
 & p\ [\,] \\
= & \text{assumption} \\
 & (\mathsf{all}\ q \cdot \mathsf{inits})\ [\,] \\
= & \text{definition of inits} \\
 & \mathsf{all}\ q\ [[\,]] \\
= & \text{definition of all} \\
 & q\ [\,] \\
= & \text{assumption} \\
 & \mathsf{True}
\end{array}
$$

and

$$
\begin{array}{ll}
 & p\ (x{+\!\!+}[a]) \\
= & \text{assumption} \\
 & (\mathsf{all}\ q \cdot \mathsf{inits})\ (x{+\!\!+}[a]) \\
= & \text{definition of inits} \\
 & \mathsf{all}\ q\ (\mathsf{inits}\ x{+\!\!+}[x{+\!\!+}[a]]) \\
= & \mathsf{all}\ q\ (x{+\!\!+}y) = \mathsf{all}\ q\ x \wedge \mathsf{all}\ q\ y \\
 & \mathsf{all}\ q\ (\mathsf{inits}\ x) \wedge \mathsf{all}\ q\ [x{+\!\!+}[a]] \\
= & \text{definition of all} \\
 & \mathsf{all}\ q\ (\mathsf{inits}\ x) \wedge q\ (x{+\!\!+}[a]) \\
= & \text{assumption} \\
 & p\ x \wedge q\ (x{+\!\!+}[a]) \\
= & \text{introduction of } \delta \text{ such that } \delta\ x\ a = q\ (x{+\!\!+}[a]) \\
 & p\ x \wedge \delta\ x\ a
\end{array}
$$

so p is prefix-closed. □

Addenda

Richard pointed out that the theorem would be a lot more elegant were it stated

Theorem 2 *A predicate p is prefix-closed iff it can be written in the form* all $q \cdot \mathsf{inits}^+$, *for some q.*

This obviates the need for q to hold for the empty sequence, and so the extra condition is unnecessary.

It occurred to me that the theorem and the proof would be even simpler if the q were replaced by p itself:

Theorem 3 *A predicate p is prefix-closed iff it can be written in the form* all $p \cdot \mathsf{inits}^+$.

The relationship between this and theorem 2 is that in the latter, there may be many qs which will work; p is just the strongest such. The weakest such q is given by

$$\begin{array}{rcl} q\ (x +\!\!+ [a]) & = & p\ x \Rightarrow \delta\ x\ a \\ & = & \neg p\ x \vee \delta\ x\ a \end{array}$$

The q used in first half of the original proof is somewhere in between the two.

The least-effort cabinet formation

Johan Jeuring
Lambert Meertens
CWI & RUU

1 Introduction

This note is motivated by the latest Dutch elections, and the formation of a government following on these elections. We derive an algorithm which minimizes the effort needed to form a government (the reader is supposed to be familiar with the Bird–Meertens formalism). This algorithm is an application of a theorem about problems which can be specified by means of

$$\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathbf{segs}.$$

The Dutch House of Representatives consists of 150 seats, occupied by nine parties. The parties can be ordered from left to right according to their political views.

It is generally acknowledged that a government should be supported by at least 80 seats from the House of Representatives. Furthermore, if a number of parties form a government, and there exists a party the political views of which are in between the political views of the parties forming a government, this party joins the parties in the formation of the government (this deviates from the current state of affairs in the Netherlands). Forming a cabinet becomes more difficult when more parties are involved in the negotiations.

The problem we consider is to minimize the effort of forming a cabinet. Suppose we represent the House of Representatives as a list of natural numbers. The length of the list is equal to the number of parties in the House, the parties are ordered according to their political views, and every natural number represents the number of seats a party occupies in the House. It follows from the above facts that a government is supported by a segment of the list, the sum of which is at least 80. Since forming a cabinet becomes more difficult when more parties are involved, we want to find the shortest segment satisfying the above requirement. The specification of our problem reads

$$\downarrow_{\#}/ \cdot \mathbf{sgs} \triangleleft \cdot \mathbf{segs},$$

where the predicate **sgs** (sum greater than seventy–nine) is defined by

$$\mathbf{sgs} = (\geq 80) \cdot +/.$$

2 A promotion theorem

In this section we prove a theorem which gives the conditions under which $\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{segs}$ equals the composition of a projection function with a left–reduction.

Using the Segment Decomposition Theorem, see [Bir87], we have

$$\begin{array}{ll} & \downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{segs} \\ = & \mathrm{SDT} \\ & \downarrow_{\#}/ \cdot (\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}) * \cdot \mathsf{inits} \end{array}$$

Note that the part $\downarrow_{\#}/ \cdot (\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}) *$ from the last formula is a homomorphism. We abbreviate $\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}$ to sp. The following theorem, which is not proved, gives a solution for problems of the form $h \cdot \mathsf{inits}$, where h is a homomorphism of a specific form. We have

Theorem 1 *(*inits*–promotion Theorem). Let h be a homomorphism $\oplus/ \cdot f*$ where f is a left–reduction $(\odot \rightarrowtail i)$. Then*

$$(h \cdot \mathsf{inits}, f) = (\otimes \rightarrowtail e),$$

where $e = (i, i)$, and $\otimes$ is defined by

$$(x, y) \otimes a = (x \oplus (y \odot a), y \odot a).$$

In order to apply the inits–promotion Theorem we want to find a left–reduction for $\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}$. The 'Theory of Lists' developped so far provides lemmas which give a left–reduction for functions of the form $\uparrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}$ provided p satisfies some conditions. Therefore, we try to express $\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}$ as $\uparrow_{\#}/ \cdot q \triangleleft \cdot \mathsf{tails}$ for some predicate q. The following three lemmas contain statements of the form

$$f = g \text{ on } \Delta.$$

By this statement we mean that $f\ d = g\ d$ for all $d \in \Delta$. Using this construct, it is possible to restrict the domain of equality of functions. Let Ξ be the set of lists which satisfy the predicate p, and let Θ be the set of of lists which satisfy $\neg\ p$ (clearly $\alpha* = \Xi \cup \Theta$). We have

Lemma 1 *Let p be a predicate such that $\neg\ p$ is postfix–closed, and define q by $q = \neg\ p \cdot \mathsf{tl}$. Then*

$$\mathsf{sp} = \uparrow_{\#}/ \cdot q \triangleleft \cdot \mathsf{tails} \text{ on } \Xi.$$

We abbreviate $\uparrow_{\#}/ \cdot q \triangleleft \cdot \mathsf{tails}$ to lq. This lemma is proved using the following two lemmas, which can be proved using induction.

Lemma 2 *Let $\neg\ p$ be a postfix–closed predicate. Then*

$$p \triangleleft \cdot \mathsf{tails} = [\,]^{\bullet} \text{ on } \Theta.$$

Lemma 3 *Let $\neg\, p$ be a postfix–closed predicate. Then*

$$\mathsf{lq} = \mathsf{id} \text{ on } \Theta.$$

Proof. We prove Lemma 1 by induction. Since $\neg\, p$ is postfix–closed, $\neg\, p\ [\,]$ holds, and hence $[\,]$ is not an element of Ξ, which proves the base case.

For the induction step, assume

$$p\ x \Rightarrow (\mathsf{sp}\ x = \mathsf{lq}\ x).$$

If $\neg\, p\ x +\!\!+ [a]$ holds, then again $[a] +\!\!+ x$ is not an element from Ξ and we are finished. Suppose $p\ [a] +\!\!+ x$ holds. By a straightforward calculation we obtain

$$\mathsf{sp}\ [a] +\!\!+ x = ([a] +\!\!+ x) \downarrow_{\#} (\mathsf{sp}\ x).$$

We distinguish two cases. First, suppose $\neg\, p\ x$ holds. Then by Lemma 2 we have

$$p \lhd \mathsf{tails}\ x = [\,],$$

and hence

$$([a] +\!\!+ x) \downarrow_{\#} (\mathsf{sp}\ x) = ([a] +\!\!+ x) \downarrow_{\#} 1_{\downarrow_{\#}} = [a] +\!\!+ x.$$

On the other hand, $q\ [a] +\!\!+ x = \neg\, p\ x$ also holds, and by applying Lemma 3 we find

$$\mathsf{lq}\ [a] +\!\!+ x = [a] +\!\!+ x.$$

Finally, if $p\ x$ holds, then

$$\begin{array}{ll} & ([a] +\!\!+ x) \downarrow_{\#} (\mathsf{sp}\ x) \\ = & \text{definition of tails, assumption} \\ & \mathsf{sp}\ x \\ = & \text{induction hypothesis} \\ & \mathsf{lq}\ x \\ = & \text{assumption} \\ & \mathsf{lq}\ [a] +\!\!+ x \end{array}$$

□

The lemmas which give a left–reduction for $\uparrow_{\#}/ \cdot p \lhd \cdot \mathsf{tails}$ all require the predicate p to be prefix–closed. Suppose $\neg\, p$ is prefix–closed. Then we have for nonempty x

$$\begin{array}{ll} & q\ x +\!\!+ [a] \\ = & \text{definition of } q \\ & \neg\, p\ \mathsf{tl}\ x +\!\!+ [a] \\ = & \neg\, p \text{ is prefix–closed} \\ & \neg\, p\ \mathsf{tl}\ x \\ = & \text{definition of } q \\ & q\ x \end{array}$$

Hence q is prefix–closed on lists of length at least two. Furthermore, $q\ [a] = \mathsf{True}$ for all elements a. Since $q\ [\,]$ is undefined, q is not prefix–closed. We call a predicate p *almost prefix–closed* if and only if it holds for all singletons and satisfies

$$p\ x +\!\!+ [a] \Rightarrow p\ x,$$

for all elements a and all nonempty lists x. The predicate q is almost prefix–closed according to the discussion above. We have the following variant of the Sliding Tails Lemma (see [BGJ89]).

Lemma 4 *Suppose p is an almost prefix-closed predicate. Then*

$$\uparrow_{\#}/ \cdot\, p \triangleleft \cdot \mathsf{tails} = (\oplus \not\rightarrow e),$$

where e is some fictitious element ω, and the operator $\oplus$ is defined by

$$x \oplus a = \begin{array}{ll} x +\!\!+ [a] & \textit{if } p\ x +\!\!+ [a] \\ (\mathsf{tl}\ x) \oplus a & \textit{if } (\neg\, p\ x +\!\!+ [a]) \wedge x \neq [\,] \\ [a] & \textit{if } x = \omega \end{array}$$

The equality given in Lemma 1 holds only for elements in Ξ. Hence sp is a left–reduction for elements in Ξ. Suppose z is an element of Θ. Then, by Lemma 3, $\mathsf{lq}\ z = z$. However, $\mathsf{sp}\ z = 1_{\downarrow_{\#}}$, as is verified by an easy calculation. We have the following equivalence

$$(\mathsf{sp}\ z = 1_{\downarrow_{\#}}) \Leftrightarrow (\neg\, p\ \mathsf{lq}\ z).$$

We define

$$x \downarrow_{\$} y = \begin{array}{ll} x & \text{if } \neg\, p\ y \\ x \downarrow_{\#} y & \text{otherwise} \end{array}$$

Note that the operator $\downarrow_{\$}$ is associative but not commutative. By the above equivalence, we have

$$\downarrow_{\#}/ \cdot (\downarrow_{\#}/ \cdot\, p \triangleleft \cdot \mathsf{tails})* \cdot \mathsf{inits} = \downarrow_{\$}/ \cdot (\uparrow_{\#}/ \cdot\, q \triangleleft \cdot \mathsf{tails})* \cdot \mathsf{inits}.$$

The following theorem is obtained by applying the inits–promotion Theorem and the variant of the Sliding Tails Lemma.

Theorem 2 *Let h be the homomorphism $\downarrow_{\#}/ \cdot\, p\triangleleft$, where $\neg\, p$ is segment–closed. Let q be the predicate $\neg\, p \cdot \mathsf{tl}$. Then*

$$h \cdot \mathsf{segs} = \pi_1 \cdot (\odot \not\rightarrow i),$$

where $i = (\omega, \omega)$, and the operator $\odot$ is defined by

$$(x, y) \odot a = (x \downarrow_{\$} (y \oplus a), y \oplus a),$$

where the operator $\oplus$ is defined by

$$x \oplus a = \begin{array}{ll} x +\!\!+ [a] & \textit{if } q\ x +\!\!+ [a] \\ (\mathsf{tl}\ x) \oplus a & \textit{if } (\neg\, q\ x +\!\!+ [a]) \wedge x \neq [\,] \\ [a] & \textit{if } x = \omega \end{array}$$

3 The solution

Consider the problem described in the introduction. It is required to find an efficient algorithm for

$$\downarrow_{\#}/ \cdot \mathbf{sgs} \lhd \cdot \mathbf{segs}.$$

The predicate **sgs** satisfies

$$\neg\ \mathbf{sgs}\ x = (+/\ x) < 80.$$

Since the list x consists of natural numbers, clearly the predicate $\neg$ **sgs** is segment–closed. We apply Theorem 2, and obtain

$$\downarrow_{\#}/ \cdot \mathbf{sgs} \lhd \cdot \mathbf{segs} = \pi_1 \cdot (\odot \not\leftarrow i),$$

where $i = (\omega, \omega)$, and the operator $\odot$ is defined by

$$(x, y) \odot a = (x \downarrow_{\mathbf{s}} (y \oplus a), y \oplus a),$$

where the operator $\oplus$ is defined by

$$x \oplus a = \begin{array}{ll} x +\!\!+ [a] & \text{if } +/\ x +\!\!+ [a] < 80 \\ (\mathrm{tl}\ x) \oplus a & \text{if } (+/\ x +\!\!+ [a] \geq 80) \wedge x \neq [\,] \\ [a] & \text{if } x = \omega \end{array}$$

This is a linear–time algorithm computing the least–effort cabinet formation.

Acknowledgements. Maarten Fokkinga and Hans Zantema made useful comments

References

[BGJ89] R.S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. *Science of Computer Programming*, 12:93–104, 1989.

[Bir87] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, pages 5–42, Springer–Verlag, 1987.

For those who love calculating:

$$\curlywedge_{\oplus} = ((+\!\!+/)\circ)\circ(\tilde{*})\circ(\circ(\tilde{\otimes}))\circ(*)$$

and

$$((*)\circ)\circ(\tilde{*}) \;=\; ((*)\circ)\circ(\circ)\circ(\circ)$$

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

The aim of Algorithmics is to derive algorithms from a specification by algebraic transformations (Meertens [4]). It is well known that bound variables tend to stand in the way of decomposing an expression into its semantic constituents, and hence hinder the calculation (Meertens [5]). Thus there is an ever growing tendency to use combinators that allow to avoid variables (see e.g., Backhouse [1]). In this note we bring it to an extreme by providing a variable free expression for cross-with-plusle, $\curlywedge_{\oplus}$. In principle the S,K,I combinators (or even a single combinator, Fokker [2]) enable us to eliminate *all* variables — but the resulting combinator code is hardly readable. (For a convincing test, run any type incorrect SASL program and inspect its error message!) Instead of S,K,I we use our old and well-known friends *compose* $\circ$, *converse* $\tilde{}$, and partial and full *sectioning* (currying):

$$\begin{aligned}(x\circ y)z &= x(yz)\\ x\tilde{\oplus}y &= y\oplus x\\ (x\oplus)y &= x\oplus y = (\oplus y)x\end{aligned}$$

$$(\oplus)x \;=\; (\oplus x)$$

As usual, function application is denoted by juxtaposition; in this note it associates *to the right*, so that $(f\circ g\circ h)x = fghx = f(g(hx))$. Recall the following defining equation for cross-with:

$$\begin{aligned}\curlywedge_{\oplus}y &= (+\!\!+/)\circ(f_y*)\\ f_y a &= (a\oplus)*y\end{aligned}$$

Notice that in the defining expression there occurs just one $+\!\!+/$, one $\oplus$, and two $*$'s. This is, of course, also true of the expression in the theorem.

Theorem

$$\curlywedge_{\oplus} \;=\; ((+\!\!+/)\circ)\circ(\tilde{*})\circ(\circ(\tilde{\otimes}))\circ(*)$$

Proof By extensionality. (In each step we underline the main part of interest.)

$$\begin{aligned}&(((+\!\!+/)\circ)\underline{\circ}(\tilde{*})\underline{\circ}(\circ(\tilde{\oplus}))\underline{\circ}(*))\;y\\ =\;& \text{three compositions applied}\\ &((+\!\!+/)\circ)\;(\tilde{*})\;(\circ(\tilde{\oplus}))\;\underline{(*)}\;y\\ =\;& \text{sectioning}\\ &((+\!\!+/)\circ)\;(\tilde{*})\;(\underline{\circ}(\tilde{\oplus}))\;(*y)\\ =\;& \text{sectioning}\end{aligned}$$

$$
\begin{array}{ll}
 & ((+\!\!+/)\circ)\ \underline{(\tilde{*})}\ ((*y)\circ(\tilde{\oplus})) \\
= & \quad \text{sectioning} \\
 & ((+\!\!+/)\circ)\ (\tilde{\underline{*}}\ ((*y)\circ(\tilde{\oplus}))) \\
= & \quad \text{converse} \\
 & ((+\!\!+/)\underline{\circ})\ (((*y)\circ(\tilde{\oplus}))*) \\
= & \quad \text{sectioning} \\
 & (+\!\!+/)\ \circ\ (((*y)\circ(\tilde{\oplus}))*)
\end{array}
$$

and so it suffices to show that $f_y = ((*y)\circ(\tilde{\oplus}))$, which we do again by extensionality:

$$
\begin{array}{ll}
 & ((*y)\underline{\circ}(\tilde{\oplus}))\ a \\
= & \quad \text{composition applied} \\
 & (*y)\ \underline{(\tilde{\oplus})}\ a \\
= & \quad \text{sectioning} \\
 & (*y)\ (\tilde{\underline{\oplus}}a) \\
= & \quad \text{converse} \\
 & (\underline{*}y)\ (a\oplus) \\
= & \quad \text{sectioning} \\
 & (a\oplus)*y
\end{array}
$$

as required. □

For those who love calculating with expressions that can hardly be understood, we give some exercises.

Exercise 1 Define $g \cdot x = gx$. What function compositions are expressed by

$$(\circ f)\circ(\oplus)\circ(g\cdot)\ ,$$

$$(h\circ)\circ(\oplus)$$

and investigate the distribution of *converse* over these.

Exercise 2 Formulate and prove — variable free — the two cross laws:

$$
\begin{array}{rcl}
f*(x\curlyvee_{\oplus}y) & = & x\curlyvee_{(f\circ)\circ(\oplus)}y \\
x\curlyvee_{(\circ g)\circ(\oplus)\circ(h\cdot)}y & = & (g*x)\curlyvee_{\oplus}(h*y)
\end{array}
$$

Exercise 3 What law is expressed by

$$((*)\circ)\circ(\tilde{*}) \ = \ ((*)\circ)\circ(\circ)\circ(\circ)$$

Exercise 4 Give an expression for cross in which also the variable $\oplus$ has been eliminated.

I have taken these problems as a case study for the question whether it is practical to identify an operation $\oplus$ with a higher order function that expects the arguments one after the other (Fokkinga [3]). As you might guess, it turns out that there are too many "shuffle" steps needed during a calculation in order to get the arguments in their place. Hence it seems better to design special operators that enable us to eliminate variables without the penalty of so many shuffle steps during the calculations. This has been done by Meertens [5].

References

[1] R.C. Backhouse. *An Exploration of the Bird-Meertens Formalism.* Technical Report CS8810, Dept of Math and Comp Sc, University of Groningen, 1988.

[2] J.D. Fokker. The construction of a 1-combinator base. May 1989. State University of Utrecht.

[3] M.M. Fokkinga. Operators as higher order functions — a case study. May 1989. CWI, Amsterdam. 11 pages.

[4] 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.

[5] L. Meertens. Constructing a calculus of programs. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*, pages 66–90, Springer Verlag, Berlin-Heidelberg-New York, 1989. LNCS 375.