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

The Squiggolist

The Squiggolist, Volume 2, Number 1, March 1991

AMSTERDAM — Shapir-Worff Hypothesis rebutted. Dr. A.P.J.M. Siebes has rebutted the well-known Shapir-Worff Hypothesis. The Shapir-Wo" " Hypot!.esis states that the thoughts of a … being depend on the language (s)… has been used by many fasci… …ior- ity. By proving t… alent to Dutch, … Shapir-Worff …

A translation from attribute grammars to catamorphisms

Maarten Fokkinga, Johan Jeuring, Lambert Meertens, and Erik Meijer

EINDHOVEN — … against Categorical … The legal proceedings agains… bers of the subversive Categorica… has started yesterday in Eindhoven. … expected that the proceedings will not la… very long, although the documents bearing on the case occupy almost a cubic metre, and consist of highly unreadable papers. The members of the Categorical Movement are charged with the horrible crime of the application of categorical methods in Computing Science. Since they categorically deny these accusations, the argumentation of the public prosecutor is expected to be clear and defini-

Utrec… In the solu… exam on Squigg… have invented a numb… member has worked for son… or contradicting the proposed … ples of the laws which do not hold … map-reduce swap:

$$+\!\!+/ \cdot f* = f* \cdot +\!\!+/$$

…skip law:

$$\cdot /id** = +\!\!+/** .$$

…tency law: for arbitrary f:

$$f \cdot f = f.$$

few studen…

Protest against perestroi- … of people in the Squig… …d a petition against … appearance of … …des ; and … last

… be bo… ing from … been presented … community. Ronald … Bear promised to do … this anarchy of nota…

Map-functor factorized

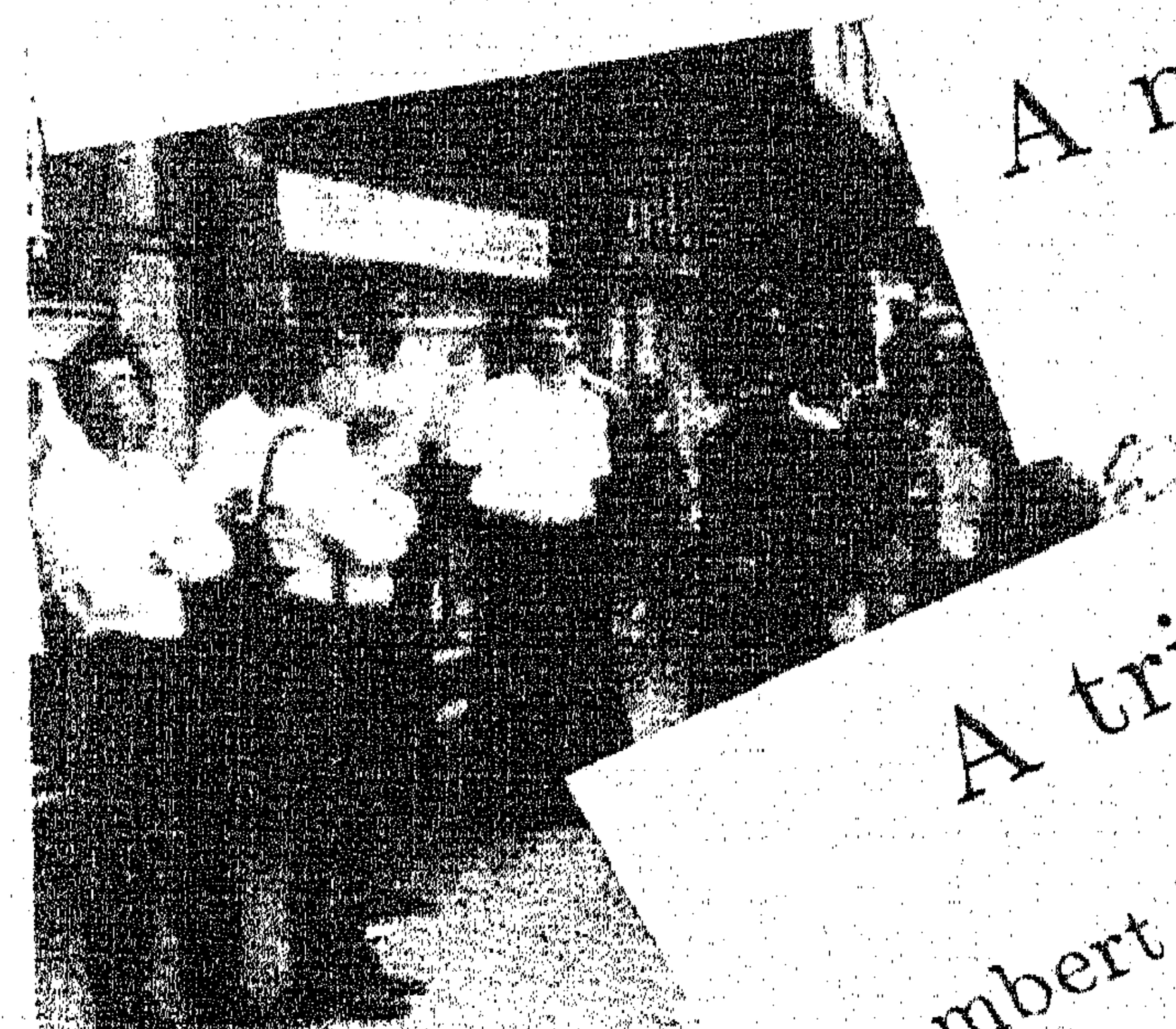
Maarten Fokkinga

From largest to maximal

Richard Bird

A neutral seggestion

Lambert Meertens



A tribute to attributes

Lambert Meertens and Jaap van der Woude

Promoting filter into cross

Norbert Völker

ZOETERMF… led by mir… admitted t… minister … day wit… …at … …

…oz- …tists … the … yester… … minister …arch w… …m … the scie…ists …romising …more … proposed to stop … s had no other re… …ed to get more?"

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

A new volume, a new colour! This is the first 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. The next issue of the Squiggolist will be produced in the summer, or, if there is a sufficient number of papers, at the end of the spring.

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

Promoting Filter into Cross

N.Völker *
University of Nijmegen

Abstract

In general, the filter operator $p \lhd$ does not promote nicely into the cross or the cartesian product operation. However, if p factorizes in certain ways, then such a promotion is possible. This holds in particular for prefix closed predicates, and in this case the resulting transformation rules can be used for the derivation of algorithms employing the so called 'list of successes technique', see e.g. [BW88].

Basic Definitions

The cross operator X takes two lists and returns a list containing all possible pairs of elements of these lists. Formally we define it by

$$\begin{aligned} x \mathsf{X} [\,] &= [\,] \\ x \mathsf{X} [a] &= (, a) * x \\ x \mathsf{X} (y +\!\!+ z) &= (x \mathsf{X} y) +\!\!+ (x \mathsf{X} z) \end{aligned}$$

On the set level, this operator coincides with the usual cartesian product. Note that we have chosen the order of the pairs to conform with [Bir87], i.e. the first index changes faster than the second index. This implies that the section $(x\ \mathsf{X})$ is a homomorphism. The relation with the 'cross with $\oplus$' operator $\mathsf{X}_\oplus$ from [Bir87] is given by the equations

$$\begin{aligned} \mathsf{X}_\oplus &= \oplus* \cdot \mathsf{X} \\ \mathsf{X} &= \mathsf{X}_{(,)} \end{aligned}$$

*This research has been supported by the Dutch organisation for scientific research NWO under grant NF 63/62 - 518

In mathematics, the cartesian product of an n-tuple of sets is a set of n-tuples. Here tuples are modelled by lists. This leads to defining the cartesian product operator cp on lists of lists (or sets) by

$$cp = \mathsf{X}_{\#} \; / \cdot \tau * *$$

where τ denotes the singleton list former function

$$\tau\, a = [a]$$

The above definition of the cp operator is typical for the so called append view of lists. However, lists can also be viewed as snoc lists, i.e. as being build up from the empty list by adding an element to the back, see e.g. [BBM+89]. We denote the *snoc* operation by $+\!\!\prec$, i.e. we set

$$x +\!\!\prec a = x +\!\!+ [a]$$

From the definition of cp, we immediately obtain the equations

$$\begin{aligned} cp\,[\,] &= [[\,]] \\ cp\,(xs +\!\!\prec x) &= cp\,xs\; \mathsf{X}_{+\!\!\prec}\; x \end{aligned}$$

which can be compressed into the following snoc world characterization of the cartesian product operator

$$cp = \mathsf{X}_{+\!\!\prec} \not\rightarrow [[\,]] \tag{1}$$

A special case of the cartesian product operation is the power operation, where taking a set x to the power n is defined as the n fold cartesian product of x with itself. Here it is expressed by the definition

$$x \uparrow n = (cp \cdot x^{\circ} *)\,[1..n]$$

where we assume $[1..0] = [\,]$ and

$$x^{\circ}\, a = x$$

i.e. x° denotes the lifting of x to a constant function.

The importance of the power operator is illustrated by the fact that the elements of $x \uparrow n$ correspond with the functions from an enumerated set of n elements into x.

Promotion Laws

Both the cross and the *cp* operator are easily proven to be natural transformations of the respective type functors. Expressed in formulas, this means that we have the following promotion laws for mapped functions into cross

$$\begin{aligned}(f\|g)* \cdot \mathsf{X} &= \mathsf{X} \cdot f* \|g* \\ f** \cdot cp &= cp \cdot f**\end{aligned}$$

We will now state two corresponding laws for filter. The proofs can be found in a separate section below.

$$(\wedge \cdot p\|q)\triangleleft \cdot \mathsf{X} = \mathsf{X} \cdot p\triangleleft \|q\triangleleft \tag{2}$$

$$(\wedge/ \cdot p*)\triangleleft \cdot cp = cp \cdot p\triangleleft * \tag{3}$$

Two special cases of the first law are worth mentioning. Let π_1 and π_2 denote the projections from a pair to its first and second element, respectively. Also, let $true = \mathrm{TRUE}^\circ$ stand for the function returning always TRUE. Using the identities

$$\begin{aligned}true\triangleleft &= id \\ \wedge \cdot p\|true &= p \cdot \pi_1 \\ \wedge \cdot true\|q &= q \cdot \pi_2\end{aligned}$$

we obtain as a direct consequence of (2)

$$(p \cdot \pi_1)\triangleleft \cdot \mathsf{X} = \mathsf{X} \cdot p\triangleleft \|id \tag{4}$$

$$(q \cdot \pi_2)\triangleleft \cdot \mathsf{X} = \mathsf{X} \cdot id\|q\triangleleft \tag{5}$$

Equation (3) gives us a promotion law for filtering with predicates of the form *all p*, where we assume the standard definition

$$all\ p = \wedge/ \cdot p*$$

This is a very strong restriction on predicates and implies that they are prefix and suffix closed. Prefix closedness on its own allows the following promotion theorem.

Theorem 1 *Let p be a prefix closed predicate and d a derivative of p, i.e. assume*

$$\begin{aligned}p\,[\,] &= TRUE \\ p(x \prec a) &= p\,x \wedge d(x, a)\end{aligned}$$

Then we have

$$p\triangleleft \cdot cp = (\prec* \cdot d\triangleleft \cdot \mathsf{X}) \not\rightarrow [[\,]] \tag{6}$$

Proof:

$$
\begin{aligned}
& p\triangleleft \cdot cp \;=\; (+\!\!\prec\!\!* \cdot d\triangleleft \cdot \mathsf{X}) \mathrel{\not\rightarrow} [[\,]] \\
\equiv\ & \{ \text{ equation (1), prefix closedness of } p \ \} \\
& p\triangleleft \cdot \mathsf{X}_{+\!\!\prec} \mathrel{\not\rightarrow} [[\,]] \;=\; (+\!\!\prec\!\!* \cdot d\triangleleft \cdot \mathsf{X}) \mathrel{\not\rightarrow} (p \triangleleft [[\,]]) \\
\Leftarrow\ & \{ \text{ snoc list promotion } \} \\
& p\triangleleft \cdot \mathsf{X}_{+\!\!\prec} \;=\; +\!\!\prec\!\!* \cdot d\triangleleft \cdot \mathsf{X} \cdot p \triangleleft \,\|id \\
\equiv\ & \{ \text{ definition of } \mathsf{X}_{+\!\!\prec}, (4) \ \} \\
& p\triangleleft \cdot +\!\!\prec\!\!* \cdot \mathsf{X} \;=\; +\!\!\prec\!\!* \cdot d\triangleleft \cdot (p \cdot \pi_1)\triangleleft \cdot \mathsf{X} \\
\equiv\ & \{ \text{ map filter swap, compositon of filters } \} \\
& +\!\!\prec\!\!* \cdot (p \cdot +\!\!\prec)\triangleleft \cdot \mathsf{X} \;=\; +\!\!\prec\!\!* \cdot (p \cdot \pi_1 \wedge d)\triangleleft \cdot \mathsf{X} \\
\equiv\ & \{ \text{ prefix closedness of } p \ \} \\
& \mathrm{TRUE}
\end{aligned}
$$

□

We leave the formulation and proof of a corresponding theorem for suffix closed predicates to the interested reader.

We will now take a look at the evaluation of the rhs of equation (6). Denote this function by f, i.e. set

$$f = (+\!\!\prec\!\!* \cdot d\triangleleft \cdot \mathsf{X}) \mathrel{\not\rightarrow} [[\,]]$$

The list $f(xs +\!\!\prec x)$ is formed by combining each element s of $f\,xs$ with those elements a of x such that $d(s,a)$ holds. In particular, only lists which satisfy predicate p are extended. This underlying idea is sometimes called the 'list of successes' technique. Note however, that if we employ outermost graph reduction as our model of computation (c.f. [BW88]), the evaluation of $f(x +\!\!\prec a)$ will be started already after the first element of the list $f\,x$ has been computed. Hence in this model of computation, f can be seen as the high level description of the classical backtracking scheme.

Theorem 1 can be generalized as follows.

Theorem 2 *Let p be a predicate, d a relation and $\oplus$ a binary operator such that*

$$p\,(x \oplus a) = p\,x \wedge d(x,a)$$

holds. Then we have

$$p\triangleleft \cdot (\mathsf{X}_{\oplus} \mathrel{\not\rightarrow} x) = (\oplus\!* \cdot d\triangleleft \cdot \mathsf{X}) \mathrel{\not\rightarrow} (p \triangleleft x)$$

Proof: From the snoc list promotion law, it follows that it is sufficient to show the validity of

$$p\lhd \cdot \mathsf{X}_{\oplus} = \oplus* \cdot d\lhd \cdot \mathsf{X} \cdot p \lhd \| id$$

However, this follows analogously to the corresponding part of the proof of theorem 1.

□

Suppose that p, d and $\oplus$ are as in theorem 2, and assume further that the binary operator $\oplus$ is associative and has a unit $1_{\oplus}$ for which p holds. Because the associativity of $\oplus$ implies the associativity of $\mathsf{X}_{\oplus}$, we can form the undirected reduce $\mathsf{X}_{\oplus}$ /. Noting the simple equation

$$1_{\mathsf{X}_{\oplus}} = [1_{\oplus}]$$

theorem 2 yields

$$p\lhd \cdot \mathsf{X}_{\oplus} / = (\oplus* \cdot d\lhd \cdot \mathsf{X}) \not\rightarrow [1_{\oplus}] \tag{7}$$

Our last promotion law deals with the special case of the power operation.

Corollary 3 *Let p be a prefix closed predicate with derivative d. Then we have*

$$p\lhd \cdot x \uparrow = f$$

where the function f is defined by

$$\begin{aligned} f\,0 &= [[\,]] \\ f\,(n+1) &= (+\!\!\prec* \cdot d\lhd)\,(f\ n\ \mathsf{X}\ x) \end{aligned}$$

Proof: It is sufficient to show that $p\lhd \cdot x \uparrow$ conforms to the same defining equations as f (Unique extension property for natural numbers!). Because prefix closed predicates hold for the empty list, the definition of the power operator $\uparrow$ implies

$$(p\lhd \cdot x \uparrow)\,0 = p \lhd [[\,]] = [[\,]]$$

As a direct consequence of theorem 1, we note

$$(p\lhd \cdot (x \uparrow))\,m = ((+\!\!\prec * \cdot d\lhd \cdot \mathsf{X}) \not\rightarrow [[\,]])\,(x^{\circ}* [1..m]) \tag{8}$$

Using this equation, the validity of the corollary follows from the calculation

$$
\begin{aligned}
&(p\lhd \cdot x\uparrow)(n+1)\\
&\qquad = \{\ \text{equation (8) for } m = n+1\ \}\\
&((+\!\!\prec * \cdot d\lhd \cdot \mathsf{X}) \not\rightarrow [[\,]])(x^\circ * [1..n+1])\\
&\qquad = \{\ \text{definition of } [1..n+1]\ \}\\
&((+\!\!\prec * \cdot d\lhd \cdot \mathsf{X}) \not\rightarrow [[\,]])(x^\circ * ([1..n] +\!\!\prec (n+1)))\\
&\qquad = \{\ h*(x +\!\!\prec a) = (h*x) +\!\!\prec (ha),\ \text{definition of } x^\circ\ \}\\
&((+\!\!\prec * \cdot d\lhd \cdot \mathsf{X}) \not\rightarrow [[\,]])((x^\circ * [1..n]) +\!\!\prec x)\\
&\qquad = \{\ \text{definition of left reduce}\ \}\\
&(+\!\!\prec * \cdot d\lhd)(((+\!\!\prec * \cdot d\lhd \cdot \mathsf{X}) \not\rightarrow [[\,]])(x^\circ * [1..n])\ \mathsf{X}\ x)\\
&\qquad = \{\ \text{equation (8) for } m = n\ \}\\
&(+\!\!\prec * \cdot d\lhd)((p\lhd \cdot x\uparrow)\, n\ \mathsf{X}\ x)
\end{aligned}
$$

□

An Example: The Eight Queens Problem

In the Eight Queens Problem (c.f. [BW88]), eight queens have to be placed on a chessboard in such a way that no two threaten each other. This obviously implies that there is exactly one queen per row, and hence we can represent the solutions by lists of length eight which contain the column positions as entries. Using the power operation introduced above, we specify

$$queens = nothreats \lhd ([1..8]\uparrow 8)$$

where the predicate *nothreats* applied to a list x is true if and only if no two queens in x threaten each other. The calculation of $nothreats(x +\!\!\prec a)$ can obviously be split into whether any two queens in x threaten each other ($nothreats\, x$) and whether the new queen (represented by a) is threatened by any of the queens in x. Giving the latter predicate the name *safe*, this means that we have the decomposition

$$nothreats(x +\!\!\prec a) = nothreats\, x \wedge safe(x, a)$$

Defining

$$nothreats\,[\,] = TRUE$$

we perceive that *nothreats* is a prefix closed predicate with derivative *safe*. Hence we can apply corollary 3 and obtain

$$queens = f\,8$$

where the function f is defined by

$$\begin{aligned} f\,0 &= [[\,]] \\ f\,(n+1) &= (+\!\!\prec * \cdot safe \lhd)\,(f\;n\;\mathsf{X}\;x) \end{aligned}$$

The remaining steps of the development depend very much on our model of computation. Assuming a functional language with outermost graph reduction and executable operators map, filter and cross, the only remaining task is the representation of the predicate *safe* by an executable function (c.f. [BW88]). If our target was the standard algorithm in either a functional language with eager evaluation or some conventional imperative language (c.f. [Wir76]), we would still have to explicitly introduce a backtracking scheme and might prefer a non executable specification of *safe*.

Proof of the cross filter promotion laws

For convenience, we recall the two equations we are going to prove

$$(\wedge \cdot p \| q) \lhd \cdot \mathsf{X} = \mathsf{X} \cdot p \lhd \| q \lhd \tag{2}$$

$$(\wedge / \cdot p*) \lhd \cdot cp = cp \cdot p \lhd * \tag{3}$$

For the proof of equation 2, note that for every list x, the left section of either side of this equation with x is $+\!\!+ - +\!\!+$ promotable. Appealing to the unique extension property, its validity therefore follows from the calculation

$$\begin{aligned} & ((\wedge \cdot p \| q) \lhd \cdot x\;\mathsf{X})\,[a]) = ((p \lhd x)\;\mathsf{X} \cdot (q \lhd)\,[a]) \\ \equiv\ & \{\ \text{operator calculus, definition of } \mathsf{X}\ \} \\ & ((\wedge \cdot p \| q) \lhd \cdot (,a)*)\,x = (\mathsf{X}\,(q \lhd [a])) \cdot p \lhd)\,x \\ \equiv\ & \{\ \text{extensionality}\ \} \\ & (\wedge \cdot p \| q) \lhd \cdot (,a)* = (\mathsf{X}\,(q \lhd [a])) \cdot p \lhd \\ \equiv\ & \{\ \text{map filter swap, case introduction}\ \} \\ & (,a)* \cdot (\wedge \cdot p \| q \cdot (,a)) \lhd = (\mathsf{X}\,(q\;a \rightarrow [a]; [\,]) \cdot p \lhd \\ \equiv\ & \{\ \text{operator calculus, case distributivity}\ \} \end{aligned}$$

$$
\begin{aligned}
& (,a)* \cdot ((\wedge(q\ a)) \cdot p)\triangleleft \;=\; (q\ a \rightarrow \mathsf{X}\,[a]; \mathsf{X}\,[]) \cdot p\triangleleft \\
\equiv\ & \{\ \text{case introduction, simplification}\ \} \\
& (,a)* \cdot (q\ a \rightarrow p; \mathit{false})\triangleleft \;=\; (q\ a \rightarrow (,a)*; \{\,\}^\circ) \cdot p\triangleleft \\
\equiv\ & \{\ \text{case distributivity, simplification}\ \} \\
& (q\ a \rightarrow (,a)* \cdot p\triangleleft; []^\circ \;=\; (q\ a \rightarrow (,a)* \cdot p\triangleleft; []^\circ \\
\equiv\ & \{\ \text{reflexivity of equality}\ \} \\
& \mathrm{TRUE}
\end{aligned}
$$

For the proof of equation 3, we denote the function on the lhs with f and the one on the rhs with g. Unfolding cp and employing map distributivity, we can write g as the compositon of a reduce and a map

$$g = \mathsf{X}_{\#}\ /\ \cdot (\tau* \cdot p\triangleleft)*$$

However, this implies trivially that we have the following equations

$$
\begin{aligned}
g\,[] &= {}^{1}\mathsf{X}_{\#} \\
g \cdot \tau &= \tau* \cdot p\triangleleft \\
g \cdot \# &= \mathsf{X}_{\#} \cdot g \| g
\end{aligned}
$$

By the unique extension property, g is on the other hand uniquely described by these equations, and it therefore suffices to show that also f conforms to these equations. This is achieved by the following straightforward calculations

$$
\begin{aligned}
f\,[] &= ((\wedge/ \cdot p*)\triangleleft \cdot cp)\,[] \\
&= (\wedge/ \cdot p*)\triangleleft\,[[\,]] \\
&= [[\,]] \\
&= {}^{1}\mathsf{X}_{\#}
\end{aligned}
$$

$$
\begin{aligned}
f \cdot \tau &= (\wedge/ \cdot p*)\triangleleft \cdot \mathsf{X}_{\#}\ /\ \cdot \tau** \cdot \tau \\
&= (\wedge/ \cdot p*)\triangleleft \cdot \mathsf{X}_{\#}\ /\ \cdot \tau \cdot \tau* \\
&= (\wedge/ \cdot p*)\triangleleft \cdot \tau* \\
&= \tau* \cdot (\wedge/ \cdot p* \cdot \tau)\triangleleft \\
&= \tau* \cdot (\wedge/ \cdot \tau \cdot p)\triangleleft \\
&= \tau* \cdot p\triangleleft
\end{aligned}
$$

$$
\begin{aligned}
f \cdot \# &= (\wedge/ \cdot p*) \triangleleft \cdot \mathsf{X}_{\#} \; / \cdot \tau** \cdot \# \\
&= (\wedge/ \cdot p*) \triangleleft \cdot \mathsf{X}_{\#} \; / \cdot \# \cdot \tau** \| \tau** \\
&= (\wedge/ \cdot p*) \triangleleft \cdot \mathsf{X}_{\#} \cdot (\mathsf{X}_{\#} \; / \cdot \tau**) \| (\mathsf{X}_{\#} \; / \cdot \tau**) \\
&= (\wedge/ \cdot p*) \triangleleft \cdot \# * \cdot \mathsf{X} \cdot (cp \| cp) \\
&= \# * \cdot (\wedge/ \cdot p* \cdot \#) \triangleleft \cdot \mathsf{X} \cdot (cp \| cp) \\
&= \# * \cdot (\wedge \cdot (\wedge/ \cdot p*) \| (\wedge/ \cdot p*)) \triangleleft \cdot \mathsf{X} \cdot (cp \| cp) \\
&\overset{eq.2}{=} \# * \cdot \mathsf{X} \cdot ((\wedge/ \cdot p*) \triangleleft \cdot cp) \| ((\wedge/ \cdot p*) \triangleleft \cdot cp) \\
&= \mathsf{X}_{\#} \cdot f \| f
\end{aligned}
$$

□

References

[BBM+89] R.S. Bird, R.C. Backhouse, G. Malcolm, O. de Moor, J.T. Jeuring, G. Jones, M.M. Fokkinga, M. Sheeran, and L.G.L.T. Meertens. International summer school on constructive algorithmics, Ameland, September 1989. Lecture notes, 1989.

[Bir87] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design. NATO ASI Series Vol. F36*, pages 5–42. Springer-Verlag, Berlin, 1987.

[BW88] R.S. Bird and P.L. Wadler. *Introduction to Functional Programming.* Prentice Hall International, Hemel Hempstead, 1988.

[Wir76] N. Wirth. *Algorithms + Datastructures = Programs.* Prentice-Hall, Englewood Cliffs, N.J., 1976.

A tribute to attributes

Lambert Meertens Jaap van der Woude

February 28, 1991

1 Introduction

A common phenomenon in programming is "flushing of results", the ingredients of which are built up in some structure (e.g. a stack). In many simple problems those ingredients are plain values and recipes to combine them (operators). This structure depends on the structure of the given data and it may be viewed as an instance of an attribute-decorated parse tree.
We shall give another example of this phenomenon where the values may be functions. The example is a stepping stone towards catamorphising attribute grammar problems in general. Two aspects of this example are especially worth mentioning:
• Parameterised algebras and their catamorphisms; we propose a general expression.
• Lifting of operators, or rather algebras. Too bad, but we have to postpone that subject.

This note depends heavily on notation and calculation techniques that can be found for instance in [2] and, in a slightly different setting, in [0]. The reader is advised to consult these references (not merely as references, they are atleast as interesting as the note here).

A remark on the priorities of the occurring operators may be in order. The general rule is that the precedence of the operator is antiproportional to its size. In weakening order we have for instance:

$$, \ . \ , \ \circ \ , \ \{\times, \vartriangle\} \ , \ \{+, \triangledown\} \ , \ \rightarrow$$

2 Depth in the leaves

The example we address may be formulated losely as follows: Given a tree, replace the values in the leaves by the depth of the given tree.
For a brief sketch of the setting of the problem:
Let $\dagger$ be the type functor for trees over A. Its unary leftsection $A\dagger$ is given by:

$$A\dagger X = A + X \times X \quad \text{and} \quad A\dagger f = id_A + f \times f$$

We fix $(Atree\ ,\ \tau_A \triangledown \shortparallel\ )$ to be the initial $A\dagger$-algebra, i.e.

$$\tau_A \triangledown \shortparallel\ : A + Atree \times Atree \longrightarrow Atree$$

The two concerns, depth and replacement, are given by the $A\dagger$-catamorphism

(1) $$\delta\ =\ (\!|\, 1^\bullet \triangledown \ddagger \,|\!) : Atree \rightarrow \mathbb{N} \quad \text{where} \quad \ddagger.(m,n)\ =\ 1 + m \uparrow n$$

and the *parameterised* $A\dagger$-catamorphism $\rho : \mathbb{N} \rightarrow (Atree \rightarrow \mathbb{N}tree)$

(2) $$\rho.n\ =\ (\!|\, (\tau_{\mathbb{N}} \triangledown \shortparallel\ ) \circ (n^\bullet + I \times I) \,|\!)\ =\ (\!|\, \tau_{\mathbb{N}} \circ n^\bullet \triangledown \shortparallel\ \,|\!)$$

In both (1) and (2) $v^\bullet$ denotes the constant v function. The function we are to express in terms of catamorphisms is $\Omega.t\ =\ \rho.(\delta.t).t$. One could defend that by (1) and (2) we are done; however, what we want as a solution is the equivalent of a one pass algorithm. In order to reformulate the requested function slightly, we need the usual apply (@) and the argument-swap "$\overline{\ }$", defined by

(3) $$\overline{\phi}.\gamma.z\ =\ \phi.z.\gamma$$

For Ω we calculate

$$\begin{array}{l} \quad \Omega.t \\ = \{\ \text{def. } \Omega\ \} \\ \quad \rho.(\delta.t).t \\ = \{\ (3)\ \} \\ \quad \overline{\rho}.t.(\delta.t) \\ = \{\ @\ \} \\ \quad @.(\overline{\rho}.t\ ,\ \delta.t) \\ = \{\ \vartriangle\ \} \\ \quad (@ \circ (\overline{\rho} \vartriangle \delta)).t \end{array}$$

Hence we are looking for $\Omega\ =\ @ \circ (\overline{\rho} \vartriangle \delta)$.

Since @ is considered to be "simple", we may reformulate our aim as: express $\overline{\rho} \vartriangle \delta$ as a catamorphism. With the tupling construction ([1]) in mind, knowing that δ is a catamorphism, it is sufficient to express $\overline{\rho}$ as a catamorphism (but we want more: an explicit expression). Is there any chance of $\overline{\rho}$ being expressable as a catamorphism? By (2) and (3), the type is right

$$\overline{\rho} : Atree \rightarrow (\mathbb{N} \rightarrow \mathbb{N}tree)$$

So if $\overline{\rho}\ =\ (\!|\psi|\!)$, we know that ψ should have the type

$$\psi\ =\ \alpha \triangledown \beta : A + (\mathbb{N} \rightarrow \mathbb{N}tree) \times (\mathbb{N} \rightarrow \mathbb{N}tree) \rightarrow (\mathbb{N} \rightarrow \mathbb{N}tree)$$

where $\alpha : A \rightarrow (\mathbb{N} \rightarrow \mathbb{N}tree)$ and $\beta : (\mathbb{N} \rightarrow \mathbb{N}tree) \times (\mathbb{N} \rightarrow \mathbb{N}tree) \rightarrow (\mathbb{N} \rightarrow \mathbb{N}tree)$.

Indeed so, in the next section we show that (don't mind the notation)

(4) $$\overline{\rho}\ =\ (\!|\, \tau^\bullet \triangledown \widehat{\shortparallel}\ \,|\!)$$

Let us calculate $\overline{\rho} \vartriangle \delta$ using (4) and the tupling construction in [1]:

$$
\begin{array}{l}
\quad \overline{\rho} \vartriangle \delta \\
= \quad \{ \ (4) \ , \ (1) \ , \ [1] \ \} \\
\quad (\!| (\tau^{\bullet} \triangledown \widehat{\#}) \circ A\dagger \ll \ \vartriangle \ (1^{\bullet} \triangledown \dagger) \circ A\dagger \gg |\!) \\
= \quad \{ \ \dagger \ , \triangledown \leftrightarrow + \ \text{calc} \ \} \\
\quad (\!| (\tau^{\bullet} \triangledown \widehat{\#} \circ (\ll \times \ll)) \vartriangle (1^{\bullet} \triangledown \dagger \circ (\gg \times \gg)) |\!) \\
= \quad \{ \ \triangledown \ \text{and} \ \vartriangle \ \text{abide} \ \} \\
\quad (\!| (\tau^{\bullet} \vartriangle 1^{\bullet}) \triangledown (\widehat{\#} \circ (\ll \times \ll) \vartriangle \dagger \circ (\gg \times \gg)) |\!) \\
= \quad \{ \ \times \leftrightarrow \vartriangle \ \text{calc.} \ , \ \infty := (\ll \times \ll) \vartriangle (\gg \times \gg) \ \} \\
\quad (\!| (\tau^{\bullet} \vartriangle 1^{\bullet}) \triangledown (\widehat{\#} \times \dagger) \circ \infty |\!)
\end{array}
$$

Indeed, this is the form we expected and it is sufficiently neat. The solution for "catamorphising" Ω being:

$$\Omega \ = \ @ \circ (\!| (\tau^{\bullet} \vartriangle 1^{\bullet}) \triangledown (\widehat{\#} \times \dagger) \circ \infty |\!)$$

Where ∞ denotes the "centre-swap" $(\ll \times \ll) \vartriangle (\gg \times \gg)$.
We still have to substantiate our claim (4). Knowing it is a useful claim, we roll up our sleeves.

3 Calculation of ψ

In the calculation to follow we use the polymorphic evaluation ε (a curried form of the apply function $@$, which in the literature is called evaluation frequently; so, be warned!):

$$\varepsilon : X \to ((X \to Y) \to Y)$$

defined by $(\varepsilon.z).\gamma \ = \ @.(\gamma, z) \ = \ \gamma.z$; or, equivalently, $\varepsilon \ = \ \overline{I}$.
There is a link between the evaluation and the argument-swap, as follows

$$(5) \qquad \overline{\theta}.z \ = \ \varepsilon.z \circ \theta$$

Assume $\psi : A\dagger(\mathbb{N} \to \mathbb{N}tree) \to (\mathbb{N} \to \mathbb{N}tree)$ such that $\overline{\rho} \ = \ (\!| \psi |\!)$ or rather $\rho \ = \ \overline{(\!| \psi |\!)}$.

$$
\begin{array}{ll}
 & \quad \rho \ = \ \overline{(\!| \psi |\!)} \\
 & \equiv \ \{ \ \text{intro} \ n \ , \ \text{heading for fusion} \ \} \\
 & \quad \rho.n \ = \ \overline{(\!| \psi |\!)}.n \\
 & \equiv \ \{ \ (2) \ , \ (5) \ \} \\
 & \quad (\!| \tau \circ n^{\bullet} \triangledown \# |\!) \ = \ \varepsilon.n \circ (\!| \psi |\!) \\
 & \Leftarrow \ \{ \ \text{fusion} \ \} \\
 & \quad (\tau \circ n^{\bullet} \triangledown \#) \circ A\dagger(\varepsilon.n) \ = \ \varepsilon.n \circ \psi \\
 & \equiv \ \{ \ \dagger \ , \ \triangledown \leftrightarrow + \ \text{calc.} \ \} \\
(\P) & \quad \tau \circ n^{\bullet} \triangledown \# \circ (\varepsilon.n \times \varepsilon.n) \ = \ \varepsilon.n \circ \psi
\end{array}
$$

Let us try to express the operands in the LHS of ¶ as

$$\tau \circ n^{\bullet} \ = \ \varepsilon.n \circ \alpha \quad \text{and} \quad \# \circ (\varepsilon.n \times \varepsilon.n) \ = \ \varepsilon.n \circ \beta$$

Indeed, $(\tau \circ n^\bullet).\gamma = \tau.n = \tau^\bullet.\gamma.n = (\varepsilon.n \circ \tau^\bullet).\gamma$
More involved is the other one:

$$
\begin{array}{l}
\quad (\ \# \circ (\varepsilon.n \times \varepsilon.n)).\gamma \\
= \{\ \gamma \in (\mathbb{N} \to \mathbb{N}tree) \times (\mathbb{N} \to \mathbb{N}tree)\ ,\ \text{say}\ \ \gamma = (\gamma_0, \gamma_1)\ \} \\
\quad \# .((\varepsilon.n).\gamma_0, (\varepsilon.n).\gamma_1) \\
= \{\ \text{def.}\ \ \varepsilon\ ,\ \ \#\ \ \text{as infix operator}\ \} \\
\quad \gamma_0.n \# \gamma_1.n \\
= \{\ \widehat{\#}\ \ \text{is lifted version of}\ \ \#\ \ \} \\
\quad (\gamma_0\ \widehat{\#}\ \gamma_1).n \\
= \{\text{def.}\ \ \varepsilon\ ,\ \ \widehat{\#}\ \ \text{as prefix operator}\ \} \\
\quad (\varepsilon.n \circ \widehat{\#}\ ).\gamma
\end{array}
$$

Hence

$$
\begin{array}{l}
\quad (\P) \\
\equiv \{\ \text{above}\ \} \\
\quad (\varepsilon.n \circ \tau^\bullet)\ \triangledown\ (\varepsilon.n \circ \widehat{\#}\ ) = \varepsilon.n \circ \psi \\
\equiv \{\ \text{distribution}\ \} \\
\quad \varepsilon.n \circ (\tau^\bullet\ \triangledown\ \widehat{\#}\ ) = \varepsilon.n \circ \psi \\
\Leftarrow \{\ \text{exit}\ \ n\ \} \\
\quad \tau^\bullet\ \triangledown\ \widehat{\#} = \psi
\end{array}
$$

This completes the proof of our claim (4).

Two intriguing questions are

- Can we give a generic solution for the catamorphisation of parametrised catamorphisms? Given a suitable setting we can, as will be demonstrated in the next section.
- We used $\widehat{\#}$, the lifted version of $\#$. Do there exist a setting and a theory for lifting in general? Indeed, they do; but we don't feel it is in a presentable form yet. (Coming next in this theatre?)

4 Parametrised algebras and catamorphisms

A *parametrised* $\dagger$-algebra for some (unary) functor $\dagger$ is a map

$$\phi : Z \to (A\dagger \to A)$$

i.e. for every $z \in Z$, $\phi.z : A\dagger \to A$ is a $\dagger$-algebra. Such a $\phi.z$ induces a $\dagger$-catamorphism $(\!|\phi.z|\!) : L \to A$, where (L, in) denotes a fixed initial $\dagger$-algebra. In other words, ϕ induces a *parametrised* catamorphism

$$(\!|?|\!) \circ \phi : Z \to (L \to A)$$

Using the argument-swap "$\overline{}$", we see that

$$\overline{(\!|?|\!) \circ \phi} : L \to (Z \to A)$$

has the right type for it to be a catamorphism, say $\overline{(\!|?|\!) \circ \phi} = (\!|\psi|\!)$, where

$$\psi : (Z \to A)\dagger \to (Z \to A)$$

Let us try and calculate ψ as we did in the former sections:

$$\begin{array}{ll} & (\!|?|\!) \circ \phi = \overline{(\!|\psi|\!)} \\ \equiv & \{ \text{ intro } z \text{ , (5) } \} \\ & (\!|\phi.z|\!) = \varepsilon.z \circ (\!|\psi|\!) \\ \Leftarrow & \{ \text{ fusion } \} \\ (\P\P) & \phi.z \circ (\varepsilon.z)\dagger = \varepsilon.z \circ \psi \end{array}$$

Here we are stuck! How do we reformulate the LHS of $(\P\P)$ such that we may "exit" z? In our example $\dagger$ was polynomial and we had an expresion for it, so we could express $(\varepsilon.z)\dagger$ explicitly. I.e. there was a map $\dot{\dagger}$ such that, for every f, $\dot{\dagger}.f = f\dagger$. This seems a strange identity: a functor always has an arrow-part, why denote it differently? The question is: are we allowed to calculate with it inside the given category; or, is the arrow-part internalisable? A functor with the property that its arrow-part may be represented as an arrow itself is sometimes called *strong*, the theatre of operations should be a Cartesian Closed Category. For a discussion of strong functors and examples see [3]. Assuming we have such a setting and a strong functor, we resume our calculation

$$\begin{array}{ll} & (\P\P) \\ \equiv & \{ \dot{\dagger} \leftrightarrow \dagger \} \{ (5) \} \\ & \phi.z \circ (\dot{\dagger} \circ \varepsilon).z = \overline{\psi}.z \\ \equiv & \{ \text{ Let } \circledcirc.(f,g) = f \circ g \} \{ \vartriangle \} \\ & (\circledcirc \circ (\phi \vartriangle \dot{\dagger} \circ \varepsilon)).z = \overline{\psi}.z \\ \equiv & \{ \text{ exit } z \} \{ \text{"}\overline{}\text{" is an inversion } \} \\ & \overline{\circledcirc \circ (\phi \vartriangle \dot{\dagger} \circ \varepsilon)} = \psi \end{array}$$

This means that we have a general expression for ψ, provided that the functor is strong *and* assuming that $\circledcirc$ is an entity inside our categorical calculational system (strongness of the exponent functor!). The argument swap in the expression may be pushed inside, albeit (for the moment) using pointwise arguments:

$$\overline{\circledcirc \circ (\phi \vartriangle \dot{\dagger} \circ \varepsilon)} = \hat{@} \circ (\phi^{\bullet} \vartriangle \overline{\dot{\dagger} \circ \varepsilon})$$

here $\hat{@}$ is the lifted version of $@$: $\hat{@}.(\theta,\chi).z = @.(\theta.z, \chi.z)$. The pointwise proof is left as an easy exercise for the reader.
The extra fun of this expression is that the leftover swapped arrow $\overline{\dot{\dagger} \circ \varepsilon}$ is a natural transformation that arises from a formalisation of lifting. Lifting is all over the place!
Although we didn't need this general expression in our example, it might be useful for a general theory on attribute grammar problems still to be developed.

Acknowledgements

The depth in leaves exercise, though standard, revived after a catalyzing talk on recursion of Norbert Völker on a Wednesday meeting at the Utrecht University. The material presented here benefitted from many inspiring discussions with Maarten Fokkinga and Johan Jeuring, especially with respect to catamorphising parametrised catamorphisms.

References

[0] Backhouse, R.C. and many others, A relational theory of datatypes, Notes workshop constructive algorithmics, Hollum, 1990.

[1] Fokkinga, M.M., Tupling and mutumorphisms, Squiggolist **1**, vol 4, 81-82, 1990.

[2] Fokkinga, M.M. and E. Meijer, Program calculation properties of continuous algebras, preprint 1990.

[3] Verwer, N., Categorical semantics as a basis for program transformations, in: A.J. van de Goor(ed), Proc. SION CSN90, deel 2, 539-554, 1990.

A Neutral Suggestion

Lambert Meertens
CWI, Amsterdam, and Utrecht University

In the formalisms we attempt to create for program calculation a humble but important role is played by what has been variously called the *unit*, *identity element* or just *identity* of a (dyadic) operation. Richard Bird has used the notation $id_\oplus$ in several papers, and Roland Backhouse has introduced the notation $1_\oplus$ for this in his unpublished Exploration paper. Recently I noticed that Richard has taken to writing $0_\oplus$.

Many of the most important operations like $+\!\!+$ and $\uplus$ are more of an additive than of a multiplicative nature, and it is indeed perhaps a bit confusing to the innocent student if $1_+ = 0$. On the other hand, if the operation *is* multiplicative, like $\times$ itself, and has a zero element, then $0_\times = 1$, as Richard's new convention would have it, is equally confusing.

An identity element is also called a *neutral* element. This suggest the notation $\nu_\oplus$, from the Greek letter *nu*, the first letter of the word "neutral" (which is of Greek origin). It has the advantage that it is neutral with respect to the perceived nature of the operation. Then we have:

$$\begin{aligned}
\nu_{+\!\!+} &= [\,] \\
\nu_\uplus &= \{\!|\,|\!\} \\
\nu_\circ &= id \\
\nu_+ &= 0 \\
\nu_\times &= 1 \\
\nu_{\times_\odot} &= (\!|\nu_\odot|\!)
\end{aligned}$$

What about zeros (absorbing elements) of a dyadic operation? Turning to Greek we have the adjective *anabrotic*, which means "gobbling up" (derived from *bibrōskein*, to devour). The word is listed in Webster as meaning "corrosive". Unfortunately, unlike "neutral", this word is virtually unknown. Still, it shares its first letter with the common word "absorbing", and so we could use:

$$\begin{aligned}
\alpha_\times &= 0 \\
\alpha_\wedge &= \mathsf{false} \\
\alpha_{\times_\odot} &= \nu_\uplus
\end{aligned}$$

Map-functor Factorized

Maarten Fokkinga, CWI & UT, Lambert Meertens, CWI & RUU

February 18, 1991

It is well known that any initial data type comes equipped with a so-called map-functor. We show that any such map-functor is the composition of two functors, one of which is —closely related to— the data type functor, and the other is —closely related to— the function μ (that for any functor F yields an initial F-algebra, if it exists).

Notation

Let K be a category, and $F : K \to K$ be an endo-functor on K. Then μF denotes "the" initial F-algebra over K, if it exists. Further, $\mathcal{F}(K)$ is the category of endo-functors on K whose morphisms are, as usual, natural transformations; and $\mathcal{F}_\mu(K)$ denotes the full sub-category of $\mathcal{F}(K)$ whose objects are those functors F for which μF exists.

For mono-functors F, G and bi-functor $\dagger$ we define the composition FG by $x(FG) = (xF)G$, and we denote by $F \dagger G$ the mono-functor defined by $x(F \dagger G) = xF \dagger xG$. Object A when used as a functor is defined by $xA = A$ for any object x and $fA = id_A$ for any morphism f. (An alternative notation for $A \dagger I$ is the 'section' $A\dagger$.) In the examples we assume that $X, \grave{\pi}, \acute{\pi}, \triangle$ form a product, and $+, \grave{\iota}, \acute{\iota}, \triangledown$ a co-product.

Making μ into a functor

We define a functor $\_^\mu : \mathcal{F}_\mu(K) \to K$ that is closely related to μ, and has therefore a closely related notation. For any $F, G \in Obj(\mathcal{F}_\mu(K))$ and $\phi : F \to G$ we put

(1) F^μ = target of μF

(2) $\phi^\mu = (\!|F|\, \phi \cdot \mu G|\!) : F^\mu \to G^\mu$.

Notice that by (1) we have $\mu F : F^\mu F \to F^\mu$. (Some authors in the Squiggol community are used to define $(L, in) = (F^\mu, \mu F)$.) The instance of ϕ that has to be taken in the right-hand side of (2) is $\phi_{G^\mu} : G^\mu F \to G^\mu G$; the typing $\phi^\mu : F^\mu \to G^\mu$ is then easily verified. In order to prove that $\_^\mu$ satisfies the two other functor axioms, we present a lemma first.

(3) Lemma *For $\phi : F \to G$ and $\psi : AG \to A$,*

$$(\!|F|\, \phi \cdot \psi|\!) = (\!|\phi \cdot \mu G|\!) \cdot (\!|G|\, \psi|\!).$$

Proof (Within this proof we use the law names and notation of Fokkinga & Meijer [1].) The reader may easily verify the steps by unfolding $f : \phi \xrightarrow{\mathsf{F}} \psi$ into $\phi ; f = f \mathsf{F} ; \psi$, and using $\phi : \mathsf{F} \dot{\rightarrow} \mathsf{G} \equiv (\forall f :: f\mathsf{F} ; \phi = \phi ; f\mathsf{G}).$)

required equality

$\Leftarrow$ FUSION

$(\!|\mathsf{G}|\ \psi|\!) : \phi ; \mu\mathsf{G} \xrightarrow{\mathsf{F}} \phi ; \psi$

$\Leftarrow$ NTRF TO HOMO, $\phi : \mathsf{F} \dot{\rightarrow} \mathsf{G}$

$(\!|\mathsf{G}|\ \psi|\!) : \mu\mathsf{G} \xrightarrow{\mathsf{G}} \psi$

$\equiv$ CATA HOMO

true.

(End of proof)

It is now immediate that $\_^{\mu}$ distributes over composition. For $\phi : \mathsf{F} \dot{\rightarrow} \mathsf{G}$ and $\psi : \mathsf{G} \dot{\rightarrow} \mathsf{H}$ we have $\phi ; \psi : \mathsf{F} \dot{\rightarrow} \mathsf{H}$ and

$= (\phi ; \psi)^{\mu}$

$(\!|\mathsf{F}|\ \phi ; \psi ; \mu\mathsf{H}|\!)$

$=$ Lemma (3), noting that $\psi ; \mu\mathsf{H} : \mathsf{H}^{\mu}\mathsf{G} \rightarrow \mathsf{H}^{\mu}$

$(\!|\mathsf{F}|\ \phi ; \mu\mathsf{G}|\!) ; (\!|\mathsf{G}|\ \psi ; \mu\mathsf{H}|\!)$

$= \phi^{\mu} ; \psi^{\mu}.$

It is also clear that $id^{\mu} = id$. Thus, $\_^{\mu}$ is a functor, $\_^{\mu} : \mathcal{F}_{\mu}(K) \rightarrow K$.

(4) Remark Another corollary of the lemma is this: for $\phi : \mathsf{F} \dot{\rightarrow} \mathsf{G}$ we have that $\phi^{\mu} ; f$ is a catamorphism whenever f is a catamorphism. (The typing determines that the former is an F-catamorphism, and the latter a G-catamorphism.) □

Let us look at some $\phi : \mathsf{F} \dot{\rightarrow} \mathsf{G}$ and see what ϕ^{μ} is.

Example Probably the most simple, non-trivial, choice is $\mathsf{F}, \mathsf{G} := \mathbf{1} + AX\mathrm{I}$, $\mathbf{1} + \mathrm{I}$ and $\phi := id + \grave{\pi}$. Notice that F^{μ} = the (set L of) cons-lists and $\mu\mathsf{F} = nil \triangledown cons$, G^{μ} = the (set $\mathbb{N}$ of) naturals and $\mu\mathsf{G} = zero \triangledown suc$. We find

$$\phi^{\mu} \quad = \quad (\!|\mathsf{F}|\ id + \grave{\pi} ; zero \triangledown suc|\!) \quad = \quad size \quad : \quad L \rightarrow \mathbb{N}.$$

□

Example Another non-trivial choice is $\mathsf{F} = \mathsf{G} = A + \mathrm{I}$, so that $\mathsf{F}^{\mu} = \mathsf{G}^{\mu}$ = the (set of) non-empty binary join trees over A, and $\mu\mathsf{F} = tip \triangledown join$. Apart from the trivial $id : \mathsf{F} \dot{\rightarrow} \mathsf{G}$, we have $\phi := id + \bowtie : \mathsf{F} \dot{\rightarrow} \mathsf{G}$ where $\bowtie = \grave{\pi} \vartriangle \acute{\pi}$. We have

$$\phi^{\mu} \quad = \quad (\!|id + \bowtie ; tip \triangledown join|\!) \quad = \quad \bowtie / \quad = \quad reverse.$$

Since $\_^{\mu}$ is a functor, we have a simple proof that *reverse* is its own inverse:

$$
\begin{array}{ll}
 & \mathit{reverse};\mathit{reverse} \\
= & \quad \phi^{\mu};\phi^{\mu} \\
= & \quad \text{functor axiom} \\
 & (\phi;\phi)^{\mu} \\
= & \quad \text{easy: } \bowtie;\bowtie = id \\
 & id^{\mu} \\
= & id.
\end{array}
$$

Notice also that by Remark (4), $\mathit{reverse};f$ is a catamorphism whenever f is. □

Example Let $\dagger$ be a bi-functor and let $\mathsf{F} = A\dagger\mathrm{I}$ and $\mathsf{G} = 1\dagger\mathrm{I}$. Take $\phi = !\dagger id : A\dagger\mathrm{I} \to 1\dagger\mathrm{I}$. Then

$$\phi^{\mu} \quad = \quad (\!|A\dagger\mathrm{I}|\ !\dagger id;\mu(1\dagger\mathrm{I})|\!) \quad = \quad \mathit{shape} \quad (= \quad !\text{-map}).$$

□

Factorizing map-functors

Let $\dagger$ be any bi-functor for which $\mu(A\dagger\mathrm{I})$ exists for all A. Recall that *the map-functor induced by* $\dagger$, $\_^{\varpi}$ say, is defined by

$$
\begin{array}{lcl}
A^{\varpi} & = & \text{target of } \mu(A\dagger\mathrm{I}) \\
f^{\varpi} & = & (\!|A\dagger\mathrm{I}|\ \ f\dagger id;\mu(B\dagger\mathrm{I})|\!) \quad : \quad A^{\varpi} \to B^{\varpi}
\end{array}
$$

for $f : A \to B$. We shall now define a functor $\_^{\dagger} : K \to \mathcal{F}_{\mu}(K)$ in such a way that composed with $\_^{\mu} : \mathcal{F}_{\mu}(K) \to K$ it equals the map-functor $\_^{\varpi} : K \to K$. To this end define

$$
\begin{array}{lcl}
A^{\dagger} & = & A\dagger\mathrm{I} \\
f^{\dagger} & = & f\dagger id \quad : \quad A\dagger\mathrm{I} \to B\dagger\mathrm{I} \qquad (\text{with } (f\dagger id)_C = f\dagger id_C)
\end{array}
$$

for any $f : A \to B$. (That $f^{\dagger}$ is a natural transformation is easily verified; it also follows from laws NTRF TRIV, NTRF ID, NTRF BI-DISTR from Fokkinga & Meijer [1].) Indeed

$$
\begin{array}{lcl}
A^{\dagger\mu} & = & (A\dagger\mathrm{I})^{\mu} \quad = \quad A^{\varpi} \\
f^{\dagger\mu} & = & (f\dagger id)^{\mu} \quad = \quad (\!|A\dagger\mathrm{I}|\ \ f\dagger id;\mu(B\dagger\mathrm{I})|\!) \quad = \quad f^{\varpi}.
\end{array}
$$

So $\varpi = \dagger\mu$.

Remark It can be shown that $\_^{\dagger}$ is just $\mathit{curry}(\dagger)$. (Here $\mathit{curry}(\_)$ is the well-defined functor from the category $\mathbf{A}\times\mathbf{B}\to\mathbf{C}$ to the category $\mathbf{A}\to(\mathbf{B}\to\mathbf{C})$, where each arrow denotes a category of functors with natural transformations as morphisms.) Thus, given bi-functor $\dagger$, we can express its map-functor without further auxiliary definitions as $\mathit{curry}(\dagger)$ composed with $^{\mu}$. □

References

[1] M.M. Fokkinga and E. Meijer. Program calculation properties of continuous algebras. December 1990. CWI, Amsterdam.

A Translation from Attribute Grammars to Catamorphisms

Maarten Fokkinga, Johan Jeuring, Lambert Meertens, Erik Meijer

November 9, 1990

Let AG be an attribute grammar, with underlying context free grammar G and attribute evaluation rules A. The function that decorates —according to A— a parse tree with attribute values and then delivers the synthesized attribute value of the root node, is denoted $[\![A]\!]$. We translate G into a functor F such that any parse tree for G is an element of the initial F-algebra. The attribute evaluation rules A are translated to a function ϕ such that $(\!|\mathsf{F}|\ \phi|\!)$ is, in a precise sense, equivalent to $[\![A]\!]$.

1 The translation

We begin by fixing some terminology and notations. Let AG be an attribute grammar. We define

G	=	the underlying context free grammar of AG.
X	=	the type of the inherited attributes (explained in (2, 3)).
Y	=	the type of the synthesized attributes (explained in (2, 3)).
T	=	the set of parse trees for grammar G (explained in (4)).
A	=	the attribute evaluation rules of AG (explained in (5)).
$[\![A]\!]$	=	the function that, given $t \in T$ and $x \in X$, {decorates tree t according to rules A when the inherited attribute of the root node of t is set to x, and} yields as result the synthesized attribute value $\in Y$ of the root node of t; thus $[\![A]\!] : T \times X \to Y$. (Explained in (6).)

In this note we construct a catamorphism for AG that is equivalent (equal) to $[\![A]\!]$. (Neither $[\![A]\!]$ nor the catamorphism is intended to yield a parse tree when given an actual string. However, one can extend any AG to AG' in such a way that $[\![A']\!](t, x)$ yields the fully decorated parse tree; see Example 2.)

Plan We shall proceed as follows.

- First we show that we may assume that AG has a simple form, so that the actual translation can be formulated without too many indices and the like.

- The context free grammar G determines a functor F.
- **Lemma** T is a subset of the carrier of the initial F-algebra.
- The attribute evaluation rules A determine a function $\phi : (X \to Y)^{\mathsf{F}} \to (X \to Y)$.
- **Theorem** $[\![A]\!](t, x) = (\!|\mathsf{F}|\ \phi|\!)\ t\ x$.

We assume that we are working in the category **Set**, or in a **Set**-like category, like CPO.

Simplification of the attribute grammar For notational simplicity we make the following three assumptions, without loss of generality.

(1) Any terminal a is produced only by rules of the form $lhs \to a$ where a in the right hand side has no attributes. This can be achieved by the addition of auxiliary nonterminal symbols, say one for each terminal symbol a.

(2) Any nonterminal has precisely one inherited and one synthesized attribute. This can be achieved for any AG by tupling the inherited attributes of each nonterminal, and also the synthesized ones. Notice that a tuple may be the empty tuple $() \in \mathbf{1}$.

(3) All inherited attributes have one type X say, and all synthesized attributed have one type Y say. This can be achieved thanks to the following technique. Suppose we have types $A_0, \ldots, A_{m-1}, B_0, \ldots, B_{n-1}$ and f is a function of type $A_i \to B_j$. Then we can define $f' : A_0 + \cdots + A_{m-1} + \mathbf{1} \to B_0 + \cdots + B_{n-1} + \mathbf{1}$ by

$$\begin{array}{lcll} f' & = & f_0 \triangledown \cdots \triangledown f_n & \\ f_{i'} & = & \mathsf{inj}_n \circ !A_{i'} & \text{for } i' \neq i \\ & = & \mathsf{inj}_j \circ f & \text{for } i' = i \\ !A & = & \text{the unique fct } A \to \mathbf{1} & \text{for any } A. \end{array}$$

Function f' is equivalent to f in the following sense: $f' \circ \mathsf{inj}_i = \mathsf{inj}_j \circ f$. So, if the actual attribute types are $X_0, \ldots, X_{m-1}$ and $Y_0, \ldots, Y_{n-1}$, then we can take $X = X_0 + \cdots + X_{m-1} + \mathbf{1}$ and $Y = Y_0 + \cdots Y_{n-1} + \mathbf{1}$, and adapt AG accordingly to AG' with $G' = G$ and $[\![A']\!]\hat{}t \circ \mathsf{inj}_0 = \mathsf{inj}_0 \circ [\![A]\!]\hat{}t$ where $\hat{}$ denotes currying and X_0 and Y_0 are the attribute types of the root node.

Construction of the functor Consider an arbitrary production rule of G, say

$$p : \qquad u \to v_0 \ldots v_{n-1}.$$

Here p is just a (unique) label of the rule, and to be very precise we should have subscripted all the entries in the rule with p. Let P be the set of production rule labels of G; in the

sequel we let P be the domain over which p ranges. Rule p determines a functor F_p, and all rules together determine a functor F and an F-algebra, as follows.

F_p	$=$	$\mathbf{1}$	if $n = 1$ and v_0 is terminal; recall (1)
	$=$	$\mathsf{I}\times\ldots\times\mathsf{I}$ (n times)	otherwise. The product is $\mathbf{1}$ if $n = 0$.
F	$=$	$\sum p :: \mathsf{F}_p$	
(T', in)	$=$	the/an initial F-algebra	
in_p	$=$	$in \circ \mathsf{inj}_p \quad : T'^{\mathsf{F}_p} \to T'$	for all p
in	$=$	$\triangledown p :: in_p,$	follows from preceding lines.

(I is the identity functor, and $\mathbf{1}$ is the constant functor. Mono-functor $\mathsf{F}\times\mathsf{G}$ is defined by $x(\mathsf{F}\times\mathsf{G}) = x\mathsf{F}\times x\mathsf{G}$ for any type and function x. Similarly, $x(\sum p :: \mathsf{F}_p) = \sum p :: x\mathsf{F}_p$ for any type and function x.)

(4) **Lemma** *There exists an embedding from the parse trees of G into (T', in).*

Proof As we have not yet given a definition of (the algebra of) parse trees, we do it here. A *parse tree t for production rule p* consists of an indication "$node_p$" and n (possibly 0) immediate constituents $t_0, \ldots, t_{n-1}$ such that, for all i, t_i is a parse tree for a production rule that has v_i in its left hand side. Let us use the notation "$node_p(t_0, \ldots, t_{n-1})$" for t. Thus "$node_p$" is made into a partial operation of type $T^n \to T$. It is partial since the arguments of operation $node_p$ have to satisfy a condition.

Now define function $\epsilon : (T, \quad \triangledown p :: node_p) \to (T', \quad \triangledown p :: in_p)$ by $\epsilon(node_p(t_0, \ldots, t_{n-1})) = in_p(\epsilon t_0, \ldots, \epsilon t_{n-1})$. Thus ϵ is an F-homomorphism from T to T' that has a post-inverse; it is an embedding. (Note that ϵ does not necessarily have a pre-inverse, since not every $in_p(t_0, \ldots, t_{n-1}) \in T'$ satisfies necessarily the condition that each t_i has v_i in its root. This happens in Example 2.)

In the sequel we identify '$node_p$ with in_p. □

Attribute evaluation Consider again production rule p, now provided with the attribute evaluation rules:

$$p: \qquad u \to v_0 \cdots v_{n-1} \qquad \text{with } (f, g_0, \ldots, g_{n-1}),$$

or slightly more suggestive (the λ is explained below)

$$(5) \quad p: \qquad u(\lambda x,' f(x,y)) \quad \to \quad \cdots \; v_i(g_i(x,y), \; \lambda y_i) \; \cdots$$

where y abbreviates $(y_0, \ldots, y_{n-1})$, an abbreviation that is valid throughout the sequel. The occurrences 'λx' and 'λy_i' are binding occurrences, their scope is the entire rule and systematic renaming is allowed. This rule says, for a tree $t = in_p(t_0, \ldots, t_{n-1})$ in which every node has been assigned two values (called *attributes*), that

the second attribute of t	$=$	$f(x,y)$
the first attribute of t_i	$=$	$g_i(x,y)$ for all i

; the first attribute of t and y_i is the second attribute of t_i. Now $[\![A]\!]$ is defined
e function such that

(t, x) = Let t' be tree t in which every node has been assigned two values (called attributes) in such a way that the first attribute of the root of t' equals x and all subtrees satisfy the above condition (for the appropriate p). Then yield as result the second attribute of the root of t'.

: that AG is such that a $[\![A]\!]$ exists; in the proof of Theorem (8) we shall further
: choice for $[\![A]\!]$. The specification implies

$(in_p(t_0, \ldots, t_{n-1}), x) = f(x, y)$ where $i :: y_i = [\![A]\!](t_i, g_i(x, y))$.

ion suggests to compute the second attribute of any (sub)tree by "attribute
" within that (sub)tree; hence this attribute is called *synthesized*. Similarly,
on suggests that the first attribute of any (sub)tree is to be determined by the
e., by attribute evaluation in the enclosing tree or by the environment in case
ree is the entire parse tree; hence this attribute is called *inherited*. As argued in
is (2, 3), the typing within rule p (5) is: $x : X$, $y : Y^n$, $f : X \times Y^n \to Y$, and
$\to X$.

tion of the catamorphism Attribute grammar rule p (5) determines a func-
id all rules together determine a function ϕ and a catamorphism $(\!|\phi|\!)$ as follows.

$\ldots, \phi_{n-1}) = (\lambda x :: f(x, y)$ where $i :: y_i = \phi_i(g_i(x, y)))$ possibly $n = 0$
$: (X \to Y)^n \to X \to Y$
$= (\nabla p :: \phi_p)$
$: (X \to Y)^{\mathsf{F}} \to (X \to Y)$
$: T' \to (X \to Y)$.

t the where-clause defines y by recursion; this corresponds to the potential
in the attribute evaluation when $[\![A]\!](t, x)$ is computed as suggested by grammar

em $[\![A]\!](t, x) = (\!|\mathsf{F}|\ \phi|\!)\ t\, x$.

ty holds for all $t \in T'$, not only for $t \in T$. This is possible thanks to our
(2) that any node in a parse tree has precisely two attributes; these have not
d, and have been referred to as "the first" and "the second" attribute of the
e attribute evaluation rule (6). Also, we have assumed in (3) that all functions
accept all kinds of values (though they may return a result in the summand
g inputs). The theorem could have been formulated as $[\![A]\!]\hat{} = (\!|\phi|\!)$, where $\hat{}$
e currying operation.

induction on the structure of t. Suppose $t = in_p(t_0, \ldots, t_{n-1})$. (Notice that
$= 0$ so that $in_p : 1 \to (X \to Y)$; this covers the so-called base case.) We

$$
\begin{array}{lll}
 & (\!|\phi|\!)\ t\ x & \\
= & \quad \text{case assumption on } t & \\
 & (\!|\phi|\!)\ (in_p(t_0,\ldots,t_{n-1}))\ x & \\
= & \quad \text{evaluation rule for catamorphisms} & \\
 & \phi_p\ ((\!|\phi|\!)\ t_0,\ldots,(\!|\phi|\!)\ t_{n-1})\ x & \\
= & \quad \text{unfold definition of } \phi_p & \\
 & f(x,y) \text{ where } i :: y_i = (\!|\phi|\!)\ t_i\ (g_i(x,y)) & \\
(*) \quad = & \quad \text{induction hypothesis} & \\
 & f(x,y) \text{ where } i :: y_i = [\![A]\!]\ (t_i, g_i(x,y)) & \\
= & \quad \text{attribute evaluation equation (7)} & \\
 & [\![A]\!]\ (in_p(t_0,\ldots,t_{n-1}),\ x) & \\
= & \quad \text{case assumption on } t & \\
 & [\![A]\!]\ (t,x)\,. &
\end{array}
$$

In step $(*)$ of the calculation it turns out that the circularity in the attribute evaluation and the mutual recursion in the definition of ϕ_p should be resolved in the same way. For example, if in the definition of ϕ_p the y_i are defined to be the *least* fixed points of the equations $i :: y_i = \phi_i(g_i(x,y))$, then so must specification (6) of $[\![A]\!]$ be understood. □

2 Examples

Linear parse trees Consider the following attribute grammar AG:

$$
\begin{array}{ll}
p: & u(\lambda x, fy) \to u(gx, \lambda y). \\
q: & u(\lambda x, hx) \to .
\end{array}
$$

This is a very simple example since there is no circularity at all. The parse trees are linear. Attribute evaluation of a tree t of depth n gives $(f^n \circ h \circ g^n)x$ as synthesized attribute value of the root node when its inherited attribute value is set to x. In other words, $[\![A]\!]\ (t,x) = (f^n \circ h \circ g^n)x$. Our construction of the previous section gives

$$
\begin{array}{lcll}
\mathsf{F} & =_{\iota} & \mathrm{I} + \mathbf{1} & \\
T' & \cong & \mathbf{N} & \\
\phi_p(\phi) & = & (\lambda x :: fy \text{ where } y = \phi(g(x,y))) & \text{so } \phi_p = (g \circ\!\!\to f) \\
\phi_q() & = & (\lambda x :: hx) & \text{so } \phi_q = h^\bullet \\
\phi & = & \phi_p \triangledown \phi_q & \text{so } \phi = (g \circ\!\!\to f) \triangledown h^\bullet \\
(\!|\mathsf{F}|\ \phi|\!) & : & \mathbf{N} \to (X \to Y) & \\
(\!|\phi|\!)\,t & = & (g \circ\!\!\to f) \circ \cdots \circ (g \circ\!\!\to f) \circ h & = f^n \circ h \circ g^n
\end{array}
$$

where, in the last line, t is assumed to have depth n, i.e., $t = (in_p \circ \cdots \circ in_p \circ in_q)()$.

Binary parse trees The following attribute grammar works on binary (parse) trees with numbers at the tips. The attribute evaluation yields (as synthesized attribute value) a tree of the same shape as the input parse tree t, having all tip numbers equal to the minimum tip value in t. This function has been discussed by a number of people, e.g., Bird [1], Kuiper and Swierstra [3, 4], Fokkinga [2]. We use $+\!\!+$ as join-operation of trees, $[.]$ as tip-former, $\downarrow$ as minimum-operation, and we let s, t vary over trees and k, m, n over numbers. The type of trees with numbers at the tips is denoted $\mathbf{N}*$. Let us first present the attribute grammar in the conventional form, i.e., not yet simplified.

$$
\begin{array}{lll}
p: & u(t) \rightarrow v(m, \lambda m, \lambda t). & \\
q: & v(\lambda k, m{\downarrow}n, s{+\!\!+}t) \rightarrow v(k, \lambda m, \lambda s)\ \ v(k, \lambda n, \lambda t). & \\
r_i: & v(\lambda k, i, [k]) \rightarrow i. & \text{for all numbers } i
\end{array}
$$

In rule p we see that the (synthesized) first attribute value of v is specified to be equal to the (inherited) second attribute value, and in rules q and r we see (by induction) that the second attribute value of v is specified to be the minimum of the input parse tree. So, eventually, in the third attribute value of v the required tree is delivered. (Kuiper and Swierstra [3] need *ten* lines for this grammar.)

Nonterminal u has no inherited attribute; it may considered to have a "nullary" attribute of type $\mathbf{1}$. Rather than taking $X = \mathbf{1} + \mathbf{N}$ for all inherited attributes, we give u a *dummy* inherited attribute of type $\mathbf{N}$; this avoids the introduction of many injections and inspections. Also, we give u an extra synthesized attribute, and then tuple the two synthesized attributes everywhere giving values of type $Y = \mathbf{N}\times\mathbf{N}*$; in order to avoid many projections we use "parameter matching" at the binding lambdas. Thus we get the following attribute grammar that satisfies the assumptions (1, 2, 3):

$$
\begin{array}{lll}
p: & u(\lambda k, (m, t)) \rightarrow v(m, \lambda(m, t)). & \\
q: & v(\lambda k, (m{\downarrow}n,\ s{+\!\!+}t)) \rightarrow v(k, \lambda(m, s))\ \ v(k, \lambda(n, t)). & \\
r_i: & v(\lambda k, (i, [k])) \rightarrow i. & \text{for all } i
\end{array}
$$

The construction of the catamorphism gives now the following.

$$
\begin{array}{lcll}
\mathsf{F} & = & \mathbf{I} + \mathbf{II} + (\sum i :: \mathbf{1}) & \\
T' & = & \text{the carrier of the initial } \mathsf{F}\text{-algebra} & \\
\phi_p(\phi) & = & (\lambda k :: (m, t) \text{ where } (m, t) = \phi m) & \\
\phi_q(\phi, \psi) & = & (\lambda k :: (m{\downarrow}n, s{+\!\!+}t) \text{ where } (m, s) = \phi k, \quad (n, t) = \psi k) & \\
\phi_{r_i}() & = & (\lambda k :: (i, [k])) & \text{for all } i \\
\phi & = & \phi_p \triangledown \phi_q \triangledown (\triangledown i :: \phi_{r_i}) & \\
 & : & (\mathbf{N} \rightarrow \mathbf{N}\times\mathbf{N}*)\mathsf{F} \rightarrow (\mathbf{N} \rightarrow \mathbf{N}\times\mathbf{N}*) & \\
(\!|\phi|\!) & : & T' \rightarrow (\mathbf{N} \rightarrow \mathbf{N}\times\mathbf{N}*) &
\end{array}
$$

Notice that, as you can see from the equation for F, T' not only contains binary trees (as each parse trees is), but also trees in which a node has just one immediate constituent; these trees can not result from parsing by the underlying context-free grammar. See also the discussion just following Theorem (8).

References

[1] R.S. Bird. Using circular programs to eliminate multiple traversals of data. *Acta Informatica*, 21:239–250, 1984.

[2] M.M. Fokkinga. Tupling of catamorphisms yields a catamorphism. May 1990. CWI, Amsterdam.

[3] M.F. Kuiper and S.D. Swierstra. Using attribute grammars to derive efficient functional programs. In *Computing Science in The Netherlands*, pages 39–52, Stichting Informatica Onderzoek in Nederland, SION, CWI, Amsterdam, November 1987.

[4] M.F. Kuiper. *Parallel Attribute Evaluation.* PhD thesis, Utrecht University, 1989.

From largest to maximal

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

January 28, 1991

Introduction

At the recent jubilee celebrations for Lambert Meertens I gave a talk in which, partly as a joke, I proved that the longest subsequence of a sequence is the sequence itself. Here is that proof:

$$
\begin{array}{ll}
 & \sqcup_{\#}/ \cdot \mathrm{subs} \\
= & \quad \{ \text{ definition of subs, with } U\ a = \{[\,],[a]\} \ \} \\
 & \sqcup_{\#}/ \cdot \mathsf{X}_{\#}/ \cdot U* \\
= & \quad \{ \text{ semiring lemma } \} \\
 & +\!\!+/ \cdot \sqcup_{\#}/* \cdot U* \\
= & \quad \{ \text{ map distributivity } \} \\
 & +\!\!+/ \cdot (\sqcup_{\#}/ \cdot U)* \\
= & \quad \{ \text{ definition of } U \ \} \\
 & +\!\!+/ \cdot 1_{+\!\!+}* \\
= & \quad \{ \text{ identity homomorphism on lists } \} \\
 & \mathrm{id}
\end{array}
$$

Our interest here is in the step labelled "semiring lemma". An algebra $(A, \oplus, \otimes, 0, 1)$ is a semiring if: (i) $\oplus$ is associative, commutative, and idempotent, with identity element 0; (ii) $\otimes$ is associative with identity element

1; (iii) $\otimes$ distributes over $\oplus$; (iv) 0 is the zero element of $\otimes$. The semiring lemma states that $(A, \oplus, \otimes, 0, 1)$ is a semiring if and only if

$$\oplus/ \cdot X_{\otimes}/ = \otimes/ \cdot \oplus/* \qquad \in [\{A\}] \to A$$

There are many examples of semirings, including

$$(\mathbf{Z}, \sqcup, +, -\infty, 0)$$

which arises in the maximum segment sum problem, and I claimed

$$([A], \sqcup_{\#}, +\!\!+, \omega, [])$$

The step labelled "semiring lemma" in the derivation is just the assertion that the above structure is indeed a semiring.

But, as I fully realised at the time, the claim is not quite right. There is no problem in finding a total ordering $<_{\#}$ that respects length and is such that $+\!\!+$ distributes through $\sqcup_{\#}$. For example, define $x <_{\#} y$ to be true if $\#x < \#y$ or $\#x = \#y$ and x is lexicographically less than y. It is easy to check that the semiring conditions hold, so our proof goes through. However, there are many refinements of $<_{\#}$ for which $+\!\!+$ does not distribute over $\sqcup_{\#}$. What happened to these in the proof?

There are two answers. The first is that the step labelled "semiring lemma" is valid not because of the semiring lemma, but because the identity

$$\oplus/ \cdot X_{\otimes}/ \cdot F* = \otimes/ \cdot (\oplus/ \cdot F)*$$

holds provided $\oplus$ is selective (i.e. $x \oplus y$ is either x or y) and $\otimes$ distributes over $\oplus$ at least on the range of F. More precisely, supposing $F : B \to \{A\}$, we require $\otimes$ to be associative (for the reduction $\otimes/$ to be defined over lists), $\oplus$ to be associative, commutative and idempotent (for $\oplus/$ to be defined over sets), $\oplus$ to be selective, and

$$\begin{aligned} (x \oplus y) \otimes z &= (x \otimes z) \oplus (y \otimes z) \\ z \otimes (x \oplus y) &= (z \otimes x) \oplus (z \otimes y) \end{aligned}$$

for all x, y, z in the range of F.

These conditions hold for all refinements of $\sqcup_{\#}$ in our proof because U returns sequences of different lengths. So everything is OK. Although I

knew that this was the real truth of the matter and was not particularly pleased with it, I was not that bothered either. I have recently come across similar examples where the use of an identity is valid only in the context of the whole expression. In other words, although we would very much like to describe our rules in terms of general algebraic properties, this is not always possible. This problem, which Lambert has called "squiggoling in context", is most acute when we try and express all our identities at the composition level. Phrasing an expression using functional composition means we lose the context of the particular argument to which the expression is applied, and so the context conditions which may be crucial become difficult to express.

As I said above, there is a second answer. The standard response of the new breed of squiggolists to such problems is to say that the problem goes away if one replaces our functional calculus by a relational one. Since Oege de Moor has shown that our functional constructions can be extended in an essentially unique way to relations, we can continue to use the same forms of expression as before but just interpret them as relations. The only difficulty is that some equations become inequations (and so some identities become inidentities?).

In the present context, the programme of work would be to formalise the distributivity conditions of a semiring as a property of relational operators and then see if essentially the same proof goes through. This seemed to require expressing distributivity in variable free form, so that the result could be lifted from functions to relations. Lambert and I tried this for a short while the day after my talk, but we were both tired and did not pursue it in any detail.

A good job too, for the real solution is much simpler, one that involves both relations and functions but one that does not insist on identifying the two. Actually, the only kind of relation we need is the concept of a partial ordering.

Let $\sqsubseteq$ be a partial ordering on A with least element ω. Further, let $(A, \oplus, 0)$ be a monoid in which $\oplus$ is monotonic under $\sqsubseteq$, i.e.

$$x \sqsubseteq y \;\Rightarrow\; x \oplus z \sqsubseteq y \oplus z \;\wedge\; z \oplus x \sqsubseteq z \oplus y$$

Finally, let $\mathcal{M} = \mathcal{M}(\sqsubseteq)$ be a function with type $\mathcal{M} : \{A\} \to \{A\}$ that

returns the maximal elements under $\sqsubseteq$. Under the above conditions on $\sqsubseteq$ and $\oplus$ we have

$$\mathcal{M} \cdot \mathsf{X}_{\oplus}/ = \mathcal{M} \cdot \mathsf{X}_{\oplus}/ \cdot \mathcal{M}*$$

This identity replaces the semiring lemma.

Here is my new proof of the longest subsequence problem. Writing $\mathcal{M} = \mathcal{M}(\leq_{\#})$ we have

$$\begin{array}{ll}
& \mathcal{M} \cdot \mathrm{subs} \\
= & \quad \{ \text{ definition of subs, with } U\ a = \{[\,],[a]\} \ \} \\
& \mathcal{M} \cdot \mathsf{X}_{\#}/ \cdot U* \\
= & \quad \{ \ \# \text{ is monotonic under } \leq_{\#} \ \} \\
& \mathcal{M} \cdot \mathsf{X}_{\#}/ \cdot \mathcal{M}* \cdot U* \\
= & \quad \{ \text{ map distributivity } \} \\
& \mathcal{M} \cdot \mathsf{X}_{\#}/ \cdot (\mathcal{M} \cdot U)* \\
= & \quad \{ \text{ definition of } U \text{ and } \mathcal{M} \ \} \\
& \mathcal{M} \cdot \mathsf{X}_{\#}/ \cdot (1_{\cup} \cdot 1_{\#})* \\
= & \quad \{ \text{ map distributivity } \} \\
& \mathcal{M} \cdot \mathsf{X}_{\#}/ \cdot 1_{\cup}* \cdot 1_{\#}* \\
= & \quad \{ \text{ law: } \mathsf{X}_{\oplus}/ \cdot 1_{\cup}* = 1_{\cup} \cdot \oplus/ \ \} \\
& \mathcal{M} \cdot 1_{\cup} \cdot \#/ \cdot 1_{\#}* \\
= & \quad \{ \text{ identity homomorphism on lists } \} \\
& \mathcal{M} \cdot 1_{\cup} \\
= & \quad \{ \text{ one-point rule for } \mathcal{M} \ \} \\
& 1_{\cup}
\end{array}$$

In words, we have shown that the set of maximal elements, under the partial ordering that respects length, of the set of subsequences of a sequence is a singleton set consisting of the sequence itself.

I have positive warm feelings about this proof. Of course, there is nothing really new in it; for example $\mathcal{M}$ *is* just the relational interpretation

of $\sqcup_{\#}/$. I also used the same idea about five years ago in the treatment of the paragraph problem. Nevertheless, the reasoning is conducted entirely at the function level and there is not a relation in sight.