Elements of a Relational Theory of Datatypes

Roland Backhouse and Paul Hoogendijk

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Abstract

The "Boom hierarchy" is a hierarchy of types that begins at the level of trees and includes lists, bags and sets. This hierarchy forms the basis for the calculus of total functions developed by Bird and Meertens, and which has become known as the "Bird-Meertens formalism".

This paper describes a hierarchy of types that logically precedes the Boom hierarchy. We show how the basic operators of the Bird-Meertens formalism (map, reduce and filter) can be introduced in a logical sequence by beginning with a very simple structure and successively refining that structure.

The context of this work is a relational theory of datatypes, rather than a calculus of total functions. Elements of the theory necessary to the later discussion are summarised at the beginning of the paper.

1 Introduction

This paper reports on an experiment into the design of a programming algebra. The algebra is an algebra of datatypes oriented towards the calculation of polymorphic functions and relations. Its design draws most inspiration from earlier research into theories of type in a functional setting but differs from those theories in including an element of indeterminacy. The selection of results chosen for presentation here has been made on the basis of level of correlation with the work of other members of IFIP Working Group 2.1. Other published selections from the work of the research team can be found in references [18, 19, 24, 26].

The goal of our work is to reduce a large class of type-manipulation problems to straightforward calculation. The hope is that within the next century it will become feasible to pose a large variety of such problems in school-leaving examinations alongside problems in, say, the differential calculus (with the implication that they are at the same level of difficulty). In order to achieve this goal it is vital to design a programming algebra in which the combination of economical notation with elegant programming laws is used to express powerful, fundamental concepts.

Fluidity of calculation is considerably enhanced by attention to two design considerations. The first is that the operators in one's algebra should be *total functions*: their use should not be hedged with conditions on the type of their arguments, however simple those conditions may be. The second is that calculational rules should involve a minimum of bound variables (at most four being our yardstick) and no complicated nestings of universal and/or existential quantifications.

The axiomatic form of the calculus of relations developed by De Morgan, Peirce, Schröder, Tarski and others has both these attributes par excellence as well as offering mechanisms for modelling the indeterminacy that is pervasive in programming problems. It has been chosen for these reasons as the basis for our experiment.

The contribution made in [5, 4, 3] is to extend the calculus of relations with the so-called "polynomial relators". That is, axioms are added defining a unit type, "junction" and "split" operators, and then it is shown how, via the latter two operators, disjoint sum and cartesian product are defined. Sum and product are so-called "relators" (a corruption of the categorical notion of functor), and, with these as building blocks, new relators can be constructed by composition and by the construction of fixed points.

In line with our design principles the junction and split operators are total functions: this in contrast to most category-theory-inspired theories of type where type restrictions are imposed on the corresponding operators. A consequence is that the laws in our system have a recognisably different character to the laws in other systems. Instead of global type restrictions on the variables in the laws the restrictions appear — where unavoidable — in the laws themselves. One of our experimental objectives has been to explore to what extent this would impede or enhance calculations. Our experience is that this design decision was fortunate. Only occasionally do type restrictions occur in our formulae and these act as a welcome reminder to the user of the calculus, and not as a tiresome detail. In this paper only one such type restriction occurs — in the very last theorem.

The main concern of the current paper is to compare and contrast the calculus to the so-called "Bird-Meertens Formalism". This formalism (to be more precise, our own conception of it) is a calculus of total functions based on a small number of primitives and a hierarchy of types including trees and lists. The theory was set out in an inspiring paper by Meertens [23] and has been further refined and applied in a number of papers by Bird and Meertens [9, 10, 13, 11, 14].

Essentially there are just three primitive operators in the theory — "reduce", "map" and "filter". These operators are defined at each level of a hierarchy of types called the "Boom hierarchy" [1] after H.J. Boom to whom Meertens attributes the concept.

The Boom hierarchy begins at the level of trees and subsequently specialises to lists, (finite) bags and sets. In this report we describe a hierarchy of types that logically precedes the Boom hierarchy and in which all three primitive operators of the Bird-Meertens formalism can be defined. We call the hierarchy a hierarchy of "freebies" because all types within the hierarchy are described by "free" algebras (i.e. algebras free of laws). How the Boom hierarchy itself is captured in the spec calculus is described in a companion paper [18].

Space limitations have dictated the form and content of this paper. The first eight sections prepare the reader for section 9 in which the main contribution of the paper

[1] For the record: Doaitse Swierstra appears to have been responsible for coining the name "Bird-Meertens Formalism" when he cracked a joke comparing "BMF" to "BNF" — Backus-Naur Form — at a workshop in Nijmegen in April, 1988. The name "Boom hierarchy" was suggested to Roland Backhouse by Richard Bird at the same workshop.

resides. In the former sections the basic elements of the calculus are summarised but no proofs of derived rules are given. Moreover, the axiomatisation of cartesian product has been omitted since it has no bearing on the results included in section 9. Derived rules are, however, built up in a logical order which will permit the industrious reader to verify all our assertions. (The word *industrious* must be stressed: the exercise is likely to be quite time-comsuming even for those with some fluency in the calculus of relations. Nevertheless, the exercise is well worth while particularly for those not so familiar wth the calculus.) In section 9, however, we do include all details of the calculations so that the reader may assess their merit.

2 The Calculus of Relations

2.1 Axioms

In this section we summarise the axiom system in which we conduct our calculations. For pedagogic reasons we prefer to decompose the algebra into three layers with their interfaces and two special axioms. The algebra is, nevertheless, well known and can also be found in, for example, [25].

Let $\mathcal{A}$ be a set, the elements of which are to be called *specs* (from *spec*ification). We use identifiers R, S, etc., to denote specs. On $\mathcal{A}$ we impose the structure of a complete, completely distributive, complemented lattice $(\mathcal{A}, \sqcap, \sqcup, \neg, \top\!\!\top, \bot\!\!\bot)$ where "$\sqcap$" and "$\sqcup$" are associative and idempotent binary infix operators with unit elements "$\top\!\!\top$" and "$\bot\!\!\bot$", respectively, and "$\neg$" is the unary prefix operator denoting complement (or negation). We assume familiarity with the standard definition of a lattice given above.

The second layer is the monoid structure for composition: $(\mathcal{A}, \circ, I)$ where $\circ$ is an associative binary infix operator with unit element I. The interface between these two layers is: $\circ$ is coordinatewise universally "cup-junctive", i.e. for $\mathcal{V}, \mathcal{W} \subseteq \mathcal{A}$,

$$(\sqcup\mathcal{V}) \circ (\sqcup\mathcal{W}) \quad = \quad \sqcup(V, W \;:\; V \in \mathcal{V} \wedge W \in \mathcal{W} \;:\; V \circ W)$$

The third layer is the reverse structure: $(\mathcal{A}, {}^\cup)$ where "${}^\cup$" is a unary postfix operator. The interface with the first layer is that "${}^\cup$" is an isomorphism of the lattice structure, i.e. for all $R, S \in \mathcal{A}$,

$$R^\cup \sqsupseteq S \quad \equiv \quad R \sqsupseteq S^\cup.$$

The interface with the second layer is that "${}^\cup$" is a contravariant monoid isomorphism

$$(R \circ S)^\cup \quad = \quad S^\cup \circ R^\cup.$$

To the above axioms we add the so-called *middle exchange rule* relating all three layers:

$$X \sqsupseteq R \circ Y \circ S \quad \equiv \quad \neg Y \sqsupseteq R^\cup \circ \neg X \circ S^\cup$$

Our last axiom, which is sometimes referred to as "Tarski's Rule", we call the *cone rule*:

$$\top\!\!\top \circ R \circ \top\!\!\top = \top\!\!\top \quad \equiv \quad R \neq \bot\!\!\bot$$

A model for this axiom system is the set of binary relations over some universe. The interpretations of the operators and constants is as follows: $\sqcup$, $\sqcap$, $\neg$ and $\sqsupseteq$ are interpreted as the set operators union, intersection, complement (with respect to the universal relation) and containment; $\top\!\!\top$, $\perp\!\!\perp$ and I are the universal relation, the empty relation and the identity relation; finally, $\circ$ and $^\cup$ are the familiar relational composition and converse operators.

2.2 Operator precedence

Some remarks on operator precedence are necessary to enable the reader to parse our formulae. First, operators in the metalanguage ($\equiv$, $\Leftarrow$ and $\Rightarrow$ together with $\vee$ and $\wedge$) have lower precedence than operators in the object language. Next, the operators in the object language "$=$", "$\sqsupseteq$" and "$\sqsubseteq$" all have equal precedence; so do "$\sqcup$" and "$\sqcap$"; and, the former is lower than the latter. Composition "$\circ$" has a yet higher precedence than all operators mentioned thus far. Finally, all unary operators in the object language, whether prefix or postfix, have the same precedence which is the highest of all. Parentheses will be used to disambiguate expressions where necessary.

3 The Domain Operators

3.1 Monotypes

The notion of a guard as a primitive entity in a programming language was first introduced in Dijkstra's guarded command language [15]. It is a useful notion since it is more flexible than the older, more conventional notion of a conditional statement. Its particular merit is that it introduces partiality into programs and at the same time facilitates the introduction of indeterminacy thereby streamlining the derivation of programs.

A guard acts as a filter on the domain of execution of a statement. Operationally it can be viewed as a partial skip. Mathematically, a guard is just a device that enables sets — subsets of the set of all states — to be incorporated into program statements.

In the spec calculus there are two mechanisms for viewing sets as specs, and thus modelling guards, each of which has its own merits. The first is via so-called "monotypes", the second via "conditions". Axiomatically, these have the following definitions. First: we say that spec A is a *monotype* iff $I \sqsupseteq A$. Second: we say that spec p is a *right condition* iff $p = \top\!\!\top \circ p$. The dual notion of *left* condition is obtained by reversing the positions of $\top\!\!\top$ and p in the right side of the defining equation.

In the relational model we may assume, for example, that the universe $\mathbb{U}$ contains two unequal values **true** and **false**. The *monotype* boolean is then defined to be the relation

$$\{(\mathbf{true}, \mathbf{true}), (\mathbf{false}, \mathbf{false})\}$$

The *right condition* boolean is the relation

$$\{(x, \mathbf{true}), (x, \mathbf{false}) \mid x \in \mathbb{U}\}$$

It is clear that for any given universe $\mathbb{U}$ there is a one-to-one correspondence between the subsets of $\mathbb{U}$ and the monotypes. Specifically, the set A is represented by the monotype $\underline{A}$ where $x\underline{A}y \;\equiv\; x = y \in A$. Equally clear is the existence of a one-to-one correspondence between the subsets of $\mathbb{U}$ and the right conditions on $\mathbb{U}$. That is, if A is some set then the right condition defined by A is that relation A_r such that for all x and y, $xA_ry \;\equiv\; y \in A$. Similarly, the left condition corresponding to A is that relation A_l such that for all x and y, $xA_ly \;\equiv\; x \in A$.

Using monotypes to represent subsets of $\mathbb{U}$ as specs, a guard on a spec is modelled by composition of the spec, either on the left or on the right, with such a monotype. Thus, if R and S are specs and A is a monotype then $A \circ R$ and $S \circ A$ are both specs, the first being spec R after restricting elements in its left domain to those in A, and the second being the spec S after restricting elements in its right domain to those in A. Using conditions a guard on the left domain of spec R is modelled by the intersection of R with a left condition, and a guard on the right domain of R by its intersection with a right condition. In principle, this poses a dilemma in the choice of representation of guards in the spec calculus. Should one choose monotypes or conditions?

We choose monotypes, there being several reasons for doing so. One is the simple fact that guarding both on the left and on the right of a spec is accomplished in one go with monotypes whereas demanding two sorts of conditions (left and right conditions). Moreover, monotypes have very simple and convenient properties. Specifically, for all monotypes A and B

(1) $\quad A \;=\; I \sqcap A \;=\; A^{\cup} \;=\; A \circ A$

(2) $\quad A \circ B \;=\; B \circ A \;=\; A \sqcap B$

The most compelling reason, however, for choosing to represent sets by monotypes is the dominant position occupied by composition among programming primitives. Introducing a guard in the middle of a sequential composition of specs is a frequent activity that is easy to express in terms of monotypes but difficult to express with conditions.

Nevertheless conditions do have their place from time to time. They too have attractive calculational properties. In particular, they form a sublattice of the spec lattice. (That is they are closed under $\sqcup$, $\sqcap$ and $\neg$.) However, from the above it is clear that there is a one-to-one correspondence between monotypes and both types of condition which we document formally below. Exploitation of this correspondence is central to many calculations in the spec calculus.

3.2 Left and Right Domains

We need to refer to the "domain" and "co-domain" (or "range") of a spec. In order to avoid unhelpful operational interpretations we use the terms *left-domain* and *right-domain* instead. These are denoted by "<" and ">", respectively. In the context of the present paper we mainly have use for the right domain and only occasionally the left domain. Properties of the latter are therefore omitted. They can easily be discovered by dualising the properties of the right domain.

The right domain operator is defined by two conditions. First, the right domain of a spec is a monotype: for all specs R,

(3) $\qquad I \sqsupseteq R{>}$

Second, the right domain operator is one adjoint of a Galois connection between the lattice of all specs and the sublattice of the monotypes: For all specs R and monotypes A,

(4) $\quad A \sqsupseteq R{>} \quad \equiv \quad \top\!\!\top \circ A \sqsupseteq R$

(The existence of such an operator involves a non-trivial proof.) According to a general theorem on Galois connections it follows that the right domain operator is universally $\sqcup$-junctive. In particular, for all specs R and S,

(5) $\quad (R \sqcup S){>} \quad = \quad R{>} \sqcup S{>}$

An additional consequence is that the operator is monotonic.

The left domain operator is defined by

(6) $\quad R{<} \quad = \quad R^{\cup}{>}$

The one-to-one correspondence between monotypes and right conditions mentioned several times earlier is formulated precisely as follows: for all specs R,

(7) $\quad \top\!\!\top \circ R{>} \quad = \quad \top\!\!\top \circ R \quad$ and $\quad (\top\!\!\top \circ R){>} \quad = \quad R{>}$

In particular, for all right conditions p and monotypes A,

(8) $\quad \top\!\!\top \circ p{>} \quad = \quad p \quad$ and $\quad (\top\!\!\top \circ A){>} \quad = \quad A$

Relational calculus yields the following alternative definition defining $R{>}$ as the smallest monotype satisfying the equation in A, $R \circ A = R$: for all monotypes A,

(9) $\quad R \circ A = R \quad \equiv \quad A \sqsupseteq R{>}$

The following properties are also used very frequently:

(10) $\quad R{>} \quad = \quad R^{\cup}{<}$
(11) $\quad S \circ R{>} \quad = \quad \top\!\!\top \circ R \sqcap S$
(12) $\quad (R \circ S){>} \quad = \quad (R{>} \circ S){>}$
(13) $\quad (R \sqcap S \circ T){>} \quad = \quad (S^{\cup} \circ R \sqcap T){>}$

Of these properties, three are evident when specs are interpreted as relations. Two, (11) and (13), are less so. Nevertheless, it is worth drawing attention to them because they figure frequently in some of our calculations. The alternative closed form, $I \sqcap \top\!\!\top \circ R$, for $R{>}$ is obtained from (11) by instantiating S to I and simplifying.

It is immediate from (9) that

(14) $\quad R \quad = \quad R \circ R{>}$

Indeed this law is used so frequently that, after a while, we hardly bother to mention it. It follows immediately from (2) with B instantiated to A that, for all monotypes A,

(15) $\quad A = A{>}$

4 Domain Complement

For the purpose of defining conditionals (**if-then-else** statements) it is useful to have a *total* operator that has the properties of a complement operator when restricted to monotypes. We call this operator the *complemented right domain* operator.

We specify the complemented right domain of R, denoted $R{>}\!\bullet$, by the requirement that it is the greatest monotype A satisfying $\perp\!\!\!\perp \;\sqsupseteq\; R \circ A$. I.e.

(16) $\quad R{>}\!\bullet \;\sqsupseteq\; A \;\;\equiv\;\; \perp\!\!\!\perp \;\sqsupseteq\; R \circ A$

As always, such a requirement imposes on us the burden of showing that it can indeed be fulfilled. To this end we first observe several expressions equivalent to the right side of equation (16). Two of these give a closed form for $R{>}\!\bullet$ thus establishing the existence (and uniqueness) of the operator.

Lemma 17 The following are all equivalent:

(a) $\quad \perp\!\!\!\perp \;\sqsupseteq\; R \circ A$
(b) $\quad \perp\!\!\!\perp \;\sqsupseteq\; R{>} \circ A$
(c) $\quad I \sqcap \neg(R{>}) \;\sqsupseteq\; A$
(d) $\quad \perp\!\!\!\perp \;\sqsupseteq\; \top\!\!\!\top \circ R \circ A$
(e) $\quad R \;\sqsubseteq\; \neg(\top\!\!\!\top \circ A)$
(f) $\quad \neg(\top\!\!\!\top \circ R) \;\sqsupseteq\; \top\!\!\!\top \circ A$
(g) $\quad (\neg(\top\!\!\!\top \circ R)){>} \;\sqsupseteq\; A$

□

From the equivalence of (a), (c) and (g) we infer

(18) $\quad R{>}\!\bullet \;=\; I \sqcap \neg(R{>}) \;=\; (\neg(\top\!\!\!\top \circ R)){>} \quad .$

The latter two formulae are clumsy; exhibiting them serves the purpose of showing that $R{>}\!\bullet$ does exist. Both are formulae that are suggested by the intended interpretation of the complemented right domain and might have been proposed as definitions. We prefer, however, the form of (16) on the grounds that it is closer to our view of a specification and is easier to calculate with.

Several properties of the complement domain suggest themselves. Specifically:

Lemma 19

(a) $\quad R{>} \sqcup R{>}\!\bullet \;=\; I \quad$ and $\quad R{>} \sqcap R{>}\!\bullet \;=\; \perp\!\!\!\perp$
(b) $\quad R{>}\!\bullet{>}\!\bullet \;=\; R{>}$
(c) $\quad R{>}\!\bullet \;=\; R{>}\!\bullet{>}$
(d) $\quad R{>}\!\bullet \;=\; R{>}{>}\!\bullet \;=\; (\top\!\!\!\top \circ R){>}\!\bullet$
(e) $\quad R{>}\!\bullet \;\trianglelefteq\; S{>} \;\equiv\; S{>}\!\bullet \;\trianglelefteq\; R{>}$ for $\trianglelefteq \in \{\sqsubseteq, =, \sqsupseteq\}$.

□

The importance of 19(d) has to do with the fact that we have defined a *total* operator. One is tempted to make do with the complement operator in the monotype lattice — for monotype A its complement is $I \sqcap \neg A$ — or in the lattice of right (or left) conditions —

for right condition p its complement $\neg p$ in the spec lattice coincides with its complement in the lattice of right conditions. However this creates a dilemma as to which to choose, a dilemma which it is better to circumvent. Lemma 19(d) indicates that the choice is irrelevant. (We return to this matter when we introduce the definition of conditionals.)

The equivalence of (a) and (e) in lemma 17 together with the specification (16) of the complemented domain operator predict that the complemented domain operator is one adjoint of a Galois connection. It follows that the complemented domain operator is universally $\sqcup$-junctive. To be precise we have:

Theorem 20 For all sets of specs $\mathcal{V}$,

(a) $\quad (\sqcup\mathcal{V})\!>\!\bullet \;=\; \sqcap_{\mathcal{M}}\,(\mathcal{V}\!>\!\bullet)$

where $\sqcap_{\mathcal{M}}$ denotes the infimum operator in the lattice of monotypes. (I.e. $\sqcap_{\mathcal{M}}\mathcal{B} \;=\; I$ when set of monotypes $\mathcal{B}$ is empty, otherwise $\sqcap_{\mathcal{M}}\mathcal{B} \;=\; \sqcap\,\mathcal{B}$.)
In particular, for all specs R and S,

(b) $\quad (R\sqcup S)\!>\!\bullet \;=\; R\!>\!\bullet \sqcap S\!>\!\bullet$
□

In contrast, but not unexpectedly, the complemented domain operator is not universally $\sqcap$-junctive. Its $\sqcap$-junctivity properties are inextricably linked, however, to those of the normal domain operator.

Theorem 21 For all sets of specs $\mathcal{V}$,

(a) $\quad (\sqcap\mathcal{V})\!>\!\bullet \;=\; \sqcup(\mathcal{V}\!>\!\bullet) \quad\equiv\quad (\sqcap\mathcal{V})\!> \;=\; \sqcap_{\mathcal{M}}(\mathcal{V}\!>)$

In particular, for all specs R and S,

(b) $\quad (R\sqcap S)\!>\!\bullet \;=\; R\!>\!\bullet \sqcup S\!>\!\bullet \quad\equiv\quad (R\sqcap S)\!> \;=\; R\!> \sqcap S\!>$

(Note that the right side of (b) is true if R and S are both monotypes or both right conditions. These are two situations in which the lemma proves useful.)
□

4.1 Imps and Co-imps

In this subsection we define "imps" and "co-imps" as special classes of specs. As we explain immediately following definition 24, an "imp" in the relational model is a function.

Definition 22

(a) A spec f is said to be an *imp* if and only if $I \sqsupseteq f \circ f^{\cup}$.
(b) A spec f is said to be a *co-imp* if and only if $f^{\cup}$ is an imp.
(c) A spec is said to be a *bijection* if and only if it is both an imp and a co-imp.
□

We shall say that f is a bijection *to* A *from* B if it is a bijection and $f{<} = A$ and $f{>} = B$. Note that if this is the case then both A and B are monotypes and $A = f \circ f^{\cup}$ and $B = f^{\cup} \circ f$.

The intended interpretation is that an "imp" is an "imp"lementation. On the other hand, it is not the intention that all implementations are "imps". Apart from their interpretation imps have an important distributivity property not enjoyed by arbitrary specs, namely:

Theorem 23 If f is an imp then, for all non-empty sets of specs $\mathcal{V}$,

$$\sqcap(P: \ P \in \mathcal{V}: \ P) \circ f \ = \ \sqcap(P: \ P \in \mathcal{V}: \ P \circ f)$$

In particular, for all specs R and S,

$$(R \sqcap S) \circ f \ = \ (R \circ f) \sqcap (S \circ f)$$

□

Monotypes are examples of bijections. More generally, the requirement of being a function is the requirement of being single-valued on some subset of $\mathbb{U}$, the so-called "domain" of the function. The domain and range are made explicit in the following.

Definition 24 For monotypes A and B we define the set $A \longleftarrow B$ by $f \in A \longleftarrow B$ whenever

(a) $\quad A \sqsupseteq f \circ f^{\cup}$ $\qquad\qquad$ (b) $\quad f{>} = B$

The nomenclature "$f \in A \longleftarrow B$" is verbalised by saying that "f *is an imp to* A *from* B".
□

In terms of the relational model, property (24a) expresses the statement that f is *zero-* or *single-valued*, i.e. for each x there is at most one y such that $y\,\langle f\rangle\,x$, and has range A. Property (24b) expresses the statement that f is *total* on domain B, i.e. for each $x \in B$ there is at least one y such that $y\,\langle f\rangle\,x$. Their combination justifies writing "$f.x$", for each $x \in B$, denoting the unique object y in A such that $y\,\langle f\rangle\,x$. (Note that the argument x in the expression $y\,\langle f\rangle\,x$ is on the right; we view functions as relations taking input on the right to output on the left.)

We now come to the first of several *translation* rules.

Lemma 25 (Domain Translation) For all specs R and imps f, we have:

$$R{>} \circ f \ = \ f \circ (R \circ f){>}$$

□

The above domain translation rule is the embryonic form of the so-called "range translation rule" in the quantifier calculus [1]. The rule provides a mechanism for translating a restriction ($R{>}$) on the left domain of imp f into a restriction ($(R \circ f){>}$) on its right domain.

Our next goal is to show that there is also a translation rule for the complemented domain operator. Three lemmas are necessary.

Lemma 26 For all specs R and S and all imps f,

(a) $R{>\!\bullet} \circ f \;=\; f \circ (R{>\!\bullet} \circ f){>}$
(b) $S \circ (R{>\!\bullet} \circ S){>} \;\sqsupseteq\; S \circ (R \circ S){>\!\bullet}$
(c) $(R{>\!\bullet} \circ f){>} \;\sqsubseteq\; (R \circ f){>\!\bullet}$
□

Corollary 27 (Complemented-Domain Translation) For all specs R and imps f

$$f \circ (R \circ f){>\!\bullet} \;=\; R{>\!\bullet} \circ f$$

□

5 Conditionals

Conditionals (**if-then-else** statements) are, of course, a well-established feature of programming languages, and our own theory would be incomplete if they were not included. In this section we show how they are expressed and we explore in some detail their algebraic properties.

Several publications have already appeared documenting the algebraic properties of conditionals, the most comprehensive account that we know of being given by Hoare *et al* [17]. We shall therefore compare the rules given here with the list that they supply. Their notation for conditionals will also be used, its vital characteristic being that it promotes the Boolean condition to an *infix* operator. Some of the rules presented here were included in Backus's [6] Turing award lecture but his account is less comprehensive and spoiled by the choice of the multifix notation used in the language Lisp.

We take the liberty of omitting proofs about conditionals on the grounds that the properties are (or should be) unsurprising and their proofs involve only properties of the underlying lattice structure plus a few extra rules to be stated (and proven) shortly.

Definition 28 (Conditional) For all specs P we define the binary operator $\lhd P \rhd$ by:

$$R \lhd P \rhd S \;=\; R \circ P{>} \;\sqcup\; S \circ P{>\!\bullet}$$

□

The conditional $R \lhd P \rhd S$ can be viewed as a spec which applies R to those elements for which condition P holds and applies S to the other ones.

Note that conditionals are defined for *all* specs but that for all specs P, R and S,

$$R \lhd P \rhd S \;=\; R \lhd (P{>}) \rhd S \;=\; R \lhd (\top\!\!\top \circ P) \rhd S.$$

Totality of operators is something we strive for at all times: the alternative in this case would have been to restrict P either to monotypes or to right conditions. Had we done so then we would have imposed on ourselves the obligation to determine for every other operator in the calculus whether it preserves monotypes and/or right conditions. In the cases that that is not so the laws relating those operators to conditionals would inevitably have taken on much clumsier forms.

Guards are usually formed by composing primitive guards with the boolean operators. We apply the same design principle to the definition of the booleans: we seek definitions that are total on all specs but are indifferent to the choice of monotypes or right conditions as representations of sets. This leads to the following definition.

Definition 29 (Boolean Operators) The operators $\bigvee$, $\bigwedge$ and $\sim$, and constants *true* and *false* are defined by, for all sets of specs $\mathcal{P}$ and specs R,

$$
\begin{array}{llll}
\text{(a)} & \bigvee \mathcal{P} & = & (\sqcup \mathcal{P})> \\
\text{(b)} & \bigwedge \mathcal{P} & = & \sqcap_{\mathcal{M}} (\mathcal{P})> \\
\text{(c)} & \sim R & = & R{>}\bullet \\
\text{(d)} & \mathit{true} & = & I \\
\text{(e)} & \mathit{false} & = & \perp\!\!\perp
\end{array}
$$

□

Theorem 30 (Conditional Translation) For all specs R, imps f and sets (possibly empty) of specs $\mathcal{P}$, we have:

$$
\begin{array}{llll}
\text{(a)} & \bigvee \mathcal{P} \circ f & = & f \circ \bigvee(\mathcal{P} \circ f) \\
\text{(b)} & \bigwedge \mathcal{P} \circ f & = & f \circ \bigwedge(\mathcal{P} \circ f) \\
\text{(c)} & \sim R \circ f & = & f \circ \sim(R \circ f)
\end{array}
$$

Hence, for all propositional functions θ (i.e. functions from specs to specs built from the identity function, constant functions and the boolean operators $\wedge$, $\vee$, $\sim$) and all vectors of specs $\underline{P}$ of the appropriate arity,

$$
\begin{array}{llll}
\text{(d)} & \theta.\underline{P} \circ f & = & f \circ \theta.(\underline{P} \circ f)
\end{array}
$$

□

Theorem 31 The binary operator $\lhd P \rhd$ respects imps. I.e.

$$\text{imp}.(f \lhd P \rhd g) \quad \Leftarrow \quad \text{imp}.f \ \wedge \ \text{imp}.g$$

□

Theorem 31 corresponds to the theorem

$$x \ := \ E \lhd P \rhd F \quad = \quad (x \ := \ E) \ \lhd P \rhd \ (x \ := \ F)$$

in the set of properties listed by Hoare *et al* [17]. For them the most primitive implementation (thus, "imp") is an assignment and the content of their rule is that a conditional respects assignments. Their rule is thus at a lower level of abstraction than ours, and more detailed.

The theorem illustrates the sort of proof burden one encounters when type restrictions are imposed on laws. We are obliged to document this theorem because, for example, all the translation rules are restricted to translation by imps. Should we ever wish to translate a domain (say) via a conditional then we need to know in advance that the conditional is an imp.

One final lemma is necessary before we can list the laws obeyed by conditionals.

Lemma 32

$$
\begin{array}{ll}
\text{(a)} & (R \lhd P \rhd S)> = R{>} \lhd P \rhd S> \\
\text{(b)} & (R \lhd P \rhd S){>}\bullet = R{>}\bullet \lhd P \rhd S{>}\bullet
\end{array}
$$

□

The set of "unsurprising" laws that we announced earlier can now be given:

Theorem 33 For all specs P, Q, R, S, T, imps f, and non-empty set of specs $\mathcal{V}$:

(a) $R \lhd true \rhd S \;=\; R$
(b) $R \lhd false \rhd S \;=\; S$
(c) $R \lhd P \rhd R \;=\; R$
(d) $R \lhd {\sim}P \rhd S \;=\; S \lhd P \rhd R$
(e) $R \lhd P \rhd (S \lhd P \rhd T) \;=\; R \lhd P \rhd T \;=\; (R \lhd P \rhd S) \lhd P \rhd T$
(f) $R \lhd (P \wedge Q) \rhd S \;=\; (R \lhd P \rhd S) \lhd Q \rhd S$
(g) $R \lhd (P \vee Q) \rhd S \;=\; R \lhd P \rhd (R \lhd Q \rhd S)$
(h) $(\sqcup \mathcal{V}) \lhd P \rhd S \;=\; \sqcup (\mathcal{V} \lhd P \rhd S)$
(i) $(\sqcap \mathcal{V}) \lhd P \rhd S \;=\; \sqcap (\mathcal{V} \lhd P \rhd S)$
(j) $S \lhd (P \lhd Q \rhd R) \rhd T \;=\; (S \lhd P \rhd T) \lhd Q \rhd (S \lhd R \rhd T)$
(k) $(R \lhd P \rhd S) \sqcup T \;=\; (R \sqcup T) \lhd P \rhd (S \sqcup T)$
(l) $(R \lhd P \rhd S) \sqcap T \;=\; (R \sqcap T) \lhd P \rhd (S \sqcap T)$
(m) $(R \lhd P \rhd S) \lhd Q \rhd T \;=\; (R \lhd Q \rhd T) \lhd P \rhd (S \lhd Q \rhd T)$
(n) $T \circ R \lhd P \rhd S \;=\; (T \circ R) \lhd P \rhd (T \circ S)$
(o) $R \lhd P \rhd S \circ f \;=\; (R \circ f) \lhd (P \circ f) \rhd (S \circ f)$

Moreover, for all propositional functions θ and all vectors of specs $\underline{P}$ of the appropriate arity,

(p) $R \lhd \theta.\underline{P} \rhd S \circ f \;=\; (R \circ f) \lhd \theta.(\underline{P} \circ f) \rhd (S \circ f)$

□

Part (p) of this theorem is the translation rule for conditionals. Given a spec $R \lhd P \rhd S$ with right domain A and an imp $f \in A \longleftarrow B$ one may always translate it to a spec with right domain (at most) B by translating the condition at the level of its primitive components. It takes the place of the law

$$(x := E) ; (R \lhd P(x) \rhd S) \;=\; ((x := E) ; R) \lhd P(E) \rhd ((x := E) ; S)$$

in the paper by Hoare *et al* [17].

6 Relators

A fundamental argument for the use of type information in the design of large programs is that the structure of the program is governed by the structure of the data. A well-established example is the use of recursive descent to structure the parsing (and compilation) of strings defined by a context-free grammar; here the structure of the data is defined by its grammar as is the structure of the parsing program. The idea is extended in the denotational description of programming languages where a fundamental initial step is the definition of so-called domain equations; those familiar with denotational semantics know that once this step has been taken the later steps are often relatively mundane and straightforward. Users of strongly-typed languages like Pascal will argue strongly that the effective use of type declarations is extremely important for subsequent program development, and even users of untyped languages like Lisp will admit that the programming errors that they make are often caused by type violations. A fundamental goal of our research is therefore to develop calculi of program construction that lay bare the oneness of program and data structure.

In the algebraic approach to type theory the underlying principle is that a data type is a structured set of elements that is equipped with a mechanism governed by that structure for defining functions on the elements of the type. For the benefit of readers who may not be familiar with it we now outline this approach as it pertains to functional programming. A fuller account is contained in, for example, [22]. Other readers will probably wish to skip the next two paragraphs; all they need to know is that we use the term "catamorphism" to refer to F-homomorphisms whose domain is an initial F-algebra.

The approach involves several stages building up to the definition of a "universal object" in a category of algebras. First, the notion of endofunctor is of paramount importance. An endofunctor is (in this context) a pair of functions, one from types to types and the other from functions to functions. Typically, both functions are denoted by the same symbol. Suppose F is an endofunctor, A and B are types and f and g are functions of composable type. Let I_A denote the identity function on the type A. Then it is required that

$$F.f \in F.A \longleftarrow F.B \quad \Leftarrow \quad f \in A \longleftarrow B$$

$$F.I_A \quad = \quad I_{F.A}$$

and $$F.(f \circ g) \quad = \quad F.f \circ F.g$$

The next step is to define an F-algebra as a pair consisting of a type A and a function $f \in A \longleftarrow F.A$. The data type defined by the endofunctor F is then an F-algebra satisfying a so-called "universality property". Specifically, An F-algebra (A, f) is called a data type if for each F-algebra (B, g) a unique function η exists such that

$$\eta \circ f \quad = \quad g \circ F.\eta$$

To emphasise the special rôle of such homomorphisms they are given a special name, specifically the name "catamorphism".

An example would be the data type natural number. Roughly speaking, $\mathbb{N}$ has the property

$$\mathbb{N} \quad = \quad \{0\}+\mathbb{N}$$

where "+" denotes the disjoint sum of two types. (According to this definition the elements of $\mathbb{N}$ are $\hookrightarrow.0$ and $\hookleftarrow.n$ where n ranges over $\mathbb{N}$ and $\hookrightarrow$ and $\hookleftarrow$ denote the injection functions associated with disjoint sum. You should interpret "$\hookrightarrow.0$" as zero and "$\hookleftarrow$" as the successor function. More formally, we recognise in this equation an endofunctor "$\{0\}+$". This is a function that maps the type A to the type $\{0\}+A$. But it may also be extended to map functions to functions by defining $\{0\}+f$ to be that function g such that $g \circ \hookrightarrow$ is the constant function always returning $\hookrightarrow.0$, and $g \circ \hookleftarrow \; = \; \hookleftarrow \circ f$. (Moreover, it satisfies all the properties required of a functor, but that we leave to the reader to verify.) A $\{0\}+$-algebra is a set together with a constant and a unary operator (these being zero and the successor function in the case of the natural numbers), and a $\{0\}+$-homomorphism is just what one would normally understand by a homomorphism of an algebraic structure, in this case a function ϕ, say, from one $\{0\}+$-algebra $(A,(a,\sigma))$, say, to another $(B,(b,\tau))$, say, that maps the constant of the first to the constant of the second

i.e. $\quad \phi.a \;=\; b$

and commutes with the unary operator of the first replacing it with that of the second

i.e. $\quad \phi \circ \sigma \;=\; \tau \circ \phi$

That $\mathbb{N}$ is "universal" in the class of $\{0\}+$-algebras just means that for any $\{0\}+$-algebra $(A,(a,\sigma))$, say, there is a unique homomorphism mapping $\mathbb{N}$ to A. With a suitable definition of the operators it is also easily shown that $\{0\}+\mathbb{N}$ is a $\{0\}+$-algebra satisfying the universality property. Thus, $\mathbb{N}$ is a fixed point of the endofunctor $\{0\}+$ in the sense that there are homomorphisms mapping $\mathbb{N}$ to $\{0\}+\mathbb{N}$ and vice-versa which (on account of their uniqueness) are each others' inverses.

To summarise this discussion: in the framework of functional programming datatypes are fixed points of endofunctors on which are defined what we call "catamorphisms", i.e. homomorphisms satisfying a uniqueness and universality property. This is not the place to discuss the practicality of catamorphisms as a program structuring method, that being something that is addressed elsewhere. We hope however that we have provided sufficient background to motivate the definitions that follow in this section. Specifically, we explore the extension of the notion of a (functional) catamorphism to relations. For this we need the notion of "relator" instead of functor.

Definition 34 (Relator) A *relator* is a function, F, from specs to specs such that

(a) $\quad I \;\sqsupseteq\; F.I$

(b) $\quad F.R \sqsupseteq F.S \;\Leftarrow\; R \sqsupseteq S$

(c) $\quad F.(R \circ S) \;=\; F.R \circ F.S$

(d) $\quad F.(R^\cup) \;=\; (F.R)^\cup$

□

In view of (34d) we take the liberty of writing simply "$F.R^\cup$" without parentheses, thus avoiding explicit use of the property.

The above defines an *endo*relator, i.e. a unary relator from a given spec algebra $\mathcal{A}$ to itself. We also need to define a *binary* relator, i.e. a function from pairs of specs to specs. If $\otimes$ denotes a binary relator, its defining properties are as follows.

(a) $\quad I \;\sqsupseteq\; I \otimes I$

(b) $\quad R \otimes U \sqsupseteq S \otimes V \;\Leftarrow\; R \sqsupseteq S \,\wedge\, U \sqsupseteq V$

(c) $\quad (R \circ S) \otimes (U \circ V) \;=\; (R \otimes U) \circ (S \otimes V)$

(d) $\quad (R^\cup) \otimes (S^\cup) \;=\; (R \otimes S)^\cup$

The notational advantage of writing "$^\cup$" as a postfix to its argument is, of course, lost in this case.

A vital property of relators is that they commute with the domain operators.

Theorem 35 If F is a relator then

(a) $F.(R>) \;=\; (F.R)>$ (b) $F.(R<) \;=\; (F.R)<$

□

In view of theorem 35 we write "$F.R<$" and "$F.R>$" without parentheses, again in order to avoid explicit mention of the properties.

The following theorem allows a comparison to be made with our definition of "relator" and the definition of "functor" (in the category of sets).

Theorem 36 If F is a relator then

(a) A is a monotype $\Rightarrow$ $F.A$ is a monotype
(b) f is an imp $\Rightarrow$ $F.f$ is an imp
(c) f is a co-imp $\Rightarrow$ $F.f$ is a co-imp
(d) $f \in A \longleftarrow B \;\Rightarrow\; F.f \in F.A \longleftarrow F.B$

□

7 Catamorphisms

Since endorelators are, by definition, monotonic the Knaster-Tarski theorem asserts the existence of their fixed points, in particular least and greatest. Specifically, the least fixed point of the endorelator F, here denoted by μF, has the defining properties

(37) $\mu F \;=\; F.\mu F$

and, for all X,

(38) $X \;\sqsupseteq\; \mu F \;\Leftarrow\; X \;\sqsupseteq\; F.X$

We shall refer to (38) as the *induction principle*.

From the induction principle follows immediately:

Lemma 39 μF is a monotype.

□

Definition 40 For endorelator F we define a function, denoted by $(\!|F;\;\_\,|\!)$, by the properties that, for all specs R and X,

(a) $(\!|F;\;R|\!) \;=\; R \circ F.(\!|F;\;R|\!)$
(b) $X \;\sqsupseteq\; (\!|F;\;R|\!) \;\Leftarrow\; X \;\sqsupseteq\; R \circ F.X$

□

In other words, $(\!|F;\;R|\!)$ is the least solution to the equation

$$X :: \quad X \;=\; R \circ F.X$$

Its well-definedness is thus guaranteed by the Knaster-Tarski theorem.

We call specs of the form $(\!|F;\ R|\!)$ *catamorphisms* (or F*-catamorphisms* when we particularly wish to be explicit about F) and we verbalise $(\!|F;\ R|\!)$ as "(F-)catamorphism R", omitting the qualification "F" when there is no doubt about the relator in question.

For reasons that will only become clear later, we call 40(a) the *computation rule* for catamorphisms. We call 40(b) the *induction principle* for catamorphisms.

Catamorphisms can be viewed as a recursively defined specs which follow the same recursion pattern as the elements of μF.

The catamorphism $(\!|I|\!)$ is of particular importance since it is clearly the least fixed point of F. Thus, we have:

(41) $\quad \mu F \;=\; (\!|I|\!)$

The remarkable property of catamorphisms that makes them such a delight to work with is that they are the unique solution of a certain equation. Specifically, we have:

Theorem 42 (Unique Extension Property) For all specs X and R,

$$X \;=\; (\!|R|\!) \;\;\equiv\;\; X \;=\; R \circ F.X \circ \mu F$$

□

A corollary of the above that figures very prominently in program calculations is:

Corollary 43 (Catamorphism Fusion) For all $\unlhd \in \{\sqsupseteq, =, \sqsubseteq\}$,

$$U \circ (\!|V|\!) \;\unlhd\; (\!|R|\!) \;\;\Leftarrow\;\; U \circ V \;\unlhd\; R \circ F.U$$

□

Fusion laws are important. In earlier publications [2, 21] we used the term "promotion" property, this term having been used by Bird to name a technique for improving the efficiency of programs [8] and which our notion captured and generalised. Maarten Fokkinga suggested the more descriptive term "fusion" property. Several valuable properties of catamorphisms are listed below.

Theorem 44 (Monotonicity) $\quad (\!|R|\!) \sqsupseteq (\!|S|\!) \;\Leftarrow\; R \sqsupseteq S$

□

Theorem 45 (Domain Trading) For all monotypes A and specs R,

(a) $\quad (\!|R|\!) \;=\; (\!|R \circ F.A|\!) \;=\; (\!|A \circ R|\!) \;\;\Leftarrow\;\; A \sqsupseteq (R \circ F.A){<}$

(b) $\quad (\!|R|\!) \;=\; (\!|R \circ F.R{<}|\!)$

(c) $\quad (\!|R|\!) \;=\; (\!|R \circ A|\!) \;\;\Leftarrow\;\; A \sqsupseteq F.R{<}$

□

A consequence of this domain trading rule is we can now generalise (41) to a very flexible and useful form.

Theorem 46 (Identity Rule)

(a) $\quad \mu F \;=\; (\!|A|\!) \;\Leftarrow\; I \sqsupseteq A \sqsupseteq \mu F$

(b) $\quad \mu F \;=\; (\!|\mu F|\!) \;=\; (\!|I|\!) \;=\; (\!|F.I|\!)$

□

8 Sample Polynomial Relators

We have now introduced the abstract concepts of a monotype, a relator and a catamorphism but we need some concrete examples to do anything useful. For the purposes of this paper it will suffice to postulate the existence of a unit monotype and the disjoint sum relator. These and their properties are briefly discussed in the next two subsections following which we show how so-called map relators are constructed by a process of constructing endorelators and taking their fixed points. For a detailed discussion of these two classes of relators see [4, 5].

8.1 The Unit Type

The unit type (a type with exactly one element) is denoted by $\mathbf{1}$. The only two properties we demand of $\mathbf{1}$ are that it is a monotype and $\mathbf{1} \circ \top\!\!\top$ is an imp.

8.2 Disjoint Sum

To define disjoint sum we begin by postulating the existence of the two injections $\hookrightarrow$ (pronounced "inject left") and $\hookleftarrow$ (pronounced "inject right"). (Note the unconventional direction of the arrow heads. As an aid to memory, and motivation for this choice, we suggest that the reader bear in mind the diagram "$X \hookrightarrow X{+}Y \hookleftarrow Y$".) Further, we introduce two binary operators on specs, $\triangledown$ (pronounced "junc") and $+$ (pronounced "plus"), defined in terms of the projection and injection specs as follows:

$$(47)\quad P \triangledown Q \;=\; (P \circ \hookrightarrow\cup) \sqcup (Q \circ \hookleftarrow\cup)$$

$$(48)\quad P{+}Q \;=\; (\hookrightarrow \circ P) \;\triangledown\; (\hookleftarrow \circ Q)$$

The relational model that we envisage assumes that the universe is a term algebra formed by closing some base set under two unary operators $\hookrightarrow$ and $\hookleftarrow$ mapping the term x to the terms $\hookrightarrow.x$ and $\hookleftarrow.x$, respectively. The two defined operators should be familiar from their interpretations which are

$$\begin{array}{lcl} x\,\langle P \triangledown Q\rangle\, y & \equiv & \exists(z :: \; y = \hookrightarrow.z \;\wedge\; x\,\langle P\rangle\, z) \\ & & \vee\; \exists(z :: \; y = \hookleftarrow.z \;\wedge\; x\,\langle Q\rangle\, z) \\ x\,\langle P{+}Q\rangle\, y & \equiv & \exists(u,v :: \; x = \hookrightarrow.u \;\wedge\; y = \hookrightarrow.v \;\wedge\; u\,\langle P\rangle\, v) \\ & & \vee\; \exists(u,v :: \; x = \hookleftarrow.u \;\wedge\; y = \hookleftarrow.v \;\wedge\; u\,\langle Q\rangle\, v) \end{array}$$

A commonly-occurring notation for $P \triangledown Q$ is $[P,Q]$. The operators defined above have a higher precedence than composition "$\circ$".

Our first axiom is that the injections are both imps.

$$(49)\qquad I \;\sqsupseteq\; (\hookrightarrow \circ \hookrightarrow\cup) \sqcup (\hookleftarrow \circ \hookleftarrow\cup)$$

We remark that axiom (49) takes the following form when rephrased in terms of the sum operation.

$$(50)\qquad I \;\sqsupseteq\; I{+}I$$

This is reassuring since it is one step on the way to guaranteeing that + is a binary relator.

The second axiom is as follows:

$$(51)\qquad (P \triangledown Q) \circ (R \triangledown S)^{\cup} \;=\; (P \circ R^{\cup}) \sqcup (Q \circ S^{\cup})$$

It is straightforward to see that + satisfies three of the conditions necessary for it to be a relator: the first is satisfied axiomatically, and monotonicity and commutation with reverse are satisfied by construction. Distributivity with respect to composition is a special case of a "fusion" law, namely that a sum can be fused with a junc.

Theorem 52 (Junc-Sum and Sum-Sum Fusion)

(a) $\qquad (P \triangledown Q) \circ (R{+}S) \;=\; (P \circ R) \triangledown (Q \circ S)$
(b) $\qquad (P{+}Q) \circ (R{+}S) \;=\; (P \circ R) + (Q \circ S)$
□

Corollary 53 $\quad$ + is a relator.
□

One more fusion property can be added on account of the universal $\sqcup$-junctivity of composition, namely:

Theorem 54 (Spec-Junc Fusion)

$$P \circ (Q \triangledown R) \;=\; (P \circ Q) \triangledown (P \circ R)$$
□

The computation rules follow straightforwardly:

Theorem 55 (Computation Rules for Junc)

(a) $\qquad (P \triangledown Q) \circ \hookrightarrow \;=\; P$
(b) $\qquad (P \triangledown Q) \circ \hookleftarrow \;=\; Q$
In particular
(c) $\qquad (P{+}Q) \circ \hookrightarrow \;=\; \hookrightarrow \circ P$
(d) $\qquad (P{+}Q) \circ \hookleftarrow \;=\; \hookleftarrow \circ Q$
□

Theorem 56

(a) $\qquad \hookrightarrow^{\cup} \circ \hookleftarrow \;=\; \perp\!\!\!\perp \;=\; \hookleftarrow^{\cup} \circ \hookrightarrow$
(b) $\qquad \hookrightarrow^{\cup} \circ \hookrightarrow \;=\; I \;=\; \hookleftarrow^{\cup} \circ \hookleftarrow$
□

Theorem 56(b) not only predicts that the injections are co-imps but also that they are total. Formulae for the left domain of the injections are also easy to calculate:

Theorem 57

(a) $\hookrightarrow\!> \;=\; I$ and $\hookleftarrow\!> \;=\; I$
(b) $\hookrightarrow\!< \;=\; I{+}\perp\!\!\perp$ and $\hookleftarrow\!< \;=\; \perp\!\!\perp{+}I$
□

The rules for the left and right domains of junc and sum are very simple. Both domain operators distribute over sum, and over junc, but transforming the operator in one case into cup and in the other into sum.

Theorem 58

(a) $(P{+}Q)> \;=\; P> \,+\, Q>$ (b) $(P{+}Q)< \;=\; P< \,+\, Q<$
(c) $(P \triangledown Q)< \;=\; P< \,\sqcup\, Q<$ (d) $(P \triangledown Q)> \;=\; P> \,+\, Q>$
□

A possible explanation for the qualification "disjoint" in the terminology "disjoint sum" is that the two components in a junc or sum remain truly disjoint. To be precise:

Theorem 59 (Cancellation Properties) For all $\trianglelefteq \in \{\sqsubseteq, =, \sqsupseteq\}$,

(a) $P \triangledown Q \;\trianglelefteq\; R \triangledown S \;\equiv\; P \trianglelefteq R \;\wedge\; Q \trianglelefteq S$
(b) $P{+}Q \;\trianglelefteq\; R{+}S \;\equiv\; P \trianglelefteq R \;\wedge\; Q \trianglelefteq S$
□

The final theorem is that both junc and sum abide with both cup and cap.

Theorem 60 (Junc/Sum-Cup/Cap Abide Laws)

(a) $(P \triangledown Q) \sqcup (R \triangledown S) \;=\; (P \sqcup R) \triangledown (Q \sqcup S)$
(b) $(P{+}Q) \sqcup (R{+}S) \;=\; (P \sqcup R){+}(Q \sqcup S)$
(c) $(P \triangledown Q) \sqcap (R \triangledown S) \;=\; (P \sqcap R) \triangledown (Q \sqcap S)$
(d) $(P{+}Q) \sqcap (R{+}S) \;=\; (P \sqcap R){+}(Q \sqcap S)$
□

8.3 Map Relators

An important mechanism for constructing new relators is via fixed points. The relators so constructed are called *map relators* and have the following definition. (They are so named because they generalise the map function defined on lists. See the discussion following theorem 76.)

Definition 61 Suppose $\otimes$ is a binary relator. It is easy to verify that $I\otimes$ is a relator (where $(I\otimes).R \;=\; I \otimes R$). Its catamorphisms therefore exist and we may define the function ϖ from specs to specs by:

$$\varpi R \;=\; (\!|I\otimes;\;\; R \otimes I|\!)$$
□

Several properties of map relators can be obtained by instantiating the general properties of catamorphisms discussed earlier. Two of particular importance are:

Theorem 62 ϖ is a relator.
□

Theorem 63 (Map Fusion) $(\!|R|\!) \circ \varpi S \;=\; (\!|R \circ S \otimes I|\!)$

□

This concludes the overview of the calculus.

9 A Hierarchy of Freebies

9.1 The Bird-Meertens Formalism

One of the hardest tasks faced by the theoretician is the assessment of the practicality of one's work. The task is not made any easier by the immense breadth of programming problems to which any useful programming calculus should be applicable. The traditional apology for such an assessment is the presentation of a few, inevitably worn and tired, case studies. We shall not follow such a course.

The course we do follow is to pass the buck: we ask the reader not to assess the practicality of our theory but to assess the practicality of the so-called "Bird-Meertens formalism", and to combine that assessment with an evaluation of the way the formalism is rendered within our theory.

The "Bird-Meertens formalism" (to be more precise, our own conception of it) is a calculus of total functions based on a small number of primitives and a hierarchy of types including trees and lists. The theory was set out in an inspiring paper by Meertens [23] and has been further refined and applied in a number of papers by Bird and Meertens [9, 10, 13, 11, 14]. Essentially there are just three primitive operators in the theory - "reduce", "map" and "filter". (Actually, the names used by Meertens for the first two of these operators were "inserted-in" and "applied-to-all". Moreover, just the first two are primitive since filter is defined in terms of reduce and map.) These operators are defined at each level of a hierarchy of types called the "Boom hierarchy"

The basis of this hierarchy is given by what Meertens calls "D-structures". A D-structure, for given type D, is formed in one of two ways: there is an embedding function that maps an element of D into a D-structure, and there is a binary join operation that combines two D-structures into one. Thus, a D-structure is a full binary tree with elements of D at the leaves. (By "full" we mean that every interior node has exactly two children.) The embedding function and the join operation are called the *constructors* of the type. Other types in the hierarchy are obtained by adding extra algebraic structure. Trees — binary but non-full — are obtained by assuming that the base type D contains a designated **nil** element which is a left and right unit of the join operation. Lists, bags and sets are obtained by successively introducing the requirements that join is associative, symmetric and idempotent.

Meertens describes the D-structures as "about the poorest (i.e., in algebraic laws) possible algebra" and trees as "about the poorest-but-one possible algebra". Nevertheless, in this section we exploit the power of abstraction afforded by the notion of a relator to add several more levels to the Boom hierarchy each of which is "poorer" than those considered by Meertens. We call this hierarchy a hierarchy of "freebies" because all types within the hierarchy are decribed by "free" algebras (i.e. algebras free of laws). Each level is characterised by a class of relators that specialises the class at the level below it. In decreasing order of abstraction these are the "sum" relators, "grounded" and "polymorphically grounded" relators, "monadic" relators and "pointed" relators. ("Grounded" and "polymorphically grounded" relators are formally indistinguishable but it helps to introduce an artificial distinction for a first introduction.) The reason for introducing these extra levels is organisational: the goal is to pin down as clearly as possible the minimum algebraic structure necessary to be able to, first, define the three operators of the Bird-Meertens formalism and, second, establish each of the basic properties of the operators. The conciseness and systematic nature of the development about to be presented, and the fact that it can be conducted at a level yet poorer than "the poorest possible algebra" is for us the most satisfying aspect of this work.

The theorems presented in this section are more general than those in the publications of Bird and Meertens since their work is mainly restricted to total functions. A danger of generalisation is that it brings with it substantial overhead making a theory abstruse and unworkable. At this stage in our work, however, the generalisation from (total) functions to relations has been very positive bringing to mind a parallel with the extension of the domain of real numbers to complex numbers. The fact of the matter is that we are rarely aware of working with relations rather than functions. The following pages are intended to provide some justification for that claim.

9.2 Sum Relators

We begin our discussion with the so-called "sum" relators. Specifically, F is a *sum relator* if for some relators G and H and for all specs X,

$$F.X \;=\; G.X + H.X \tag{64}$$

In words, F is the (lifted) sum of G and H.

The class of sum relators is very broad but, in spite of its generality, there is surprisingly much that we can say about the class. The most important aspect of such a relator F is that we can identify the "constructors" of μF, bringing the notion of relator somewhat closer to the notion of polymorphic type as it would be defined in a conventional programming language. An additional technical aspect that proves to be very useful is that F-catamorphisms can be restricted without loss of generality to arguments that are the junc of two specs. These two aspects are considered in turn below. Throughout the remainder of this subsection we assume that equation (64) is in force.

Let us consider what consequences equation (64) has on μF. We have the following simple calculation:

$$\begin{array}{ll}
 & \mu F \\
= & \qquad \{\ \mu F \text{ is a fixpoint of } F\ \} \\
 & F.\mu F \\
= & \qquad \{\ \text{definition of } F\text{: (64)}\ \} \\
 & G.\mu F \;+\; H.\mu F \\
= & \qquad \{\ \text{definition of } +\text{: (48)}\ \} \\
 & (\hookrightarrow \circ G.\mu F) \;\;\triangledown\;\; (\hookleftarrow \circ H.\mu F)
\end{array}$$

Continuing with just the first component of this junc expression, we calculate:

$$
\begin{array}{ll}
 & \hookrightarrow \;\circ\; G.\mu F \\
= & \quad\{\ \text{computation rule: theorem 55(c)}\ \} \\
 & G.\mu F + H.\mu F \;\circ\; \hookrightarrow \\
= & \quad\{\ \text{definition of } F\text{: (64), } \mu F \;=\; F.\mu F\ \} \\
 & \mu F \;\circ\; \hookrightarrow
\end{array}
$$

Similarly,

$$\hookleftarrow \;\circ\; H.\mu F \;=\; \mu F \;\circ\; \hookleftarrow$$

Thus, introducing names τ and η for the two components of the above junc, we have established:

Theorem 65 (Constructors) For relators F, G and H such that $F \;=\; G+H$,

$$
\begin{array}{ll}
 & \mu F \;=\; \tau \triangledown \eta \\
\text{where} & \tau \;=\; \hookrightarrow \;\circ\; G.\mu F \;=\; \mu F \;\circ\; \hookrightarrow \\
\text{and} & \eta \;=\; \hookleftarrow \;\circ\; H.\mu F \;=\; \mu F \;\circ\; \hookleftarrow
\end{array}
$$

□

A paraphrase of theorem 65 might be that all elements of μF are constructed by injections of elements of $G.\mu F$ or elements of $H.\mu F$. For this reason we call τ and η the *constructors* of μF.

Note that the constructors are bijections (since they are restrictions of the two bijections $\hookrightarrow$ and $\hookleftarrow$). For their domains we have:

$$
\begin{array}{ll}
 & \tau> \\
= & \quad\{\ \text{definition of } \tau\text{: theorem 65}\ \} \\
 & (\hookrightarrow \;\circ\; G.\mu F)> \\
= & \quad\{\ \text{domains: (12)}\ \} \\
 & (\hookrightarrow> \;\circ\; G.\mu F)> \\
= & \quad\{\ \hookrightarrow> \;=\; I \ \text{: theorem 57(a)}\ \} \\
 & G.\mu F> \\
= & \quad\{\ \mu F \text{ is a monotype: (15)}\ \} \\
 & G.\mu F
\end{array}
$$

and

$$
\begin{array}{ll}
 & \tau< \\
= & \quad\{\ \text{definition of } \tau\text{: theorem 65}\ \} \\
 & (\mu F \;\circ\; \hookrightarrow)< \\
= & \quad\{\ \text{domains: dual of (12)}\ \} \\
 & (\mu F \;\circ\; \hookrightarrow<)< \\
= & \quad\{\ \hookrightarrow< \;=\; I + \perp\!\!\!\perp \ \text{: theorem 57(b)}\ \} \\
 & (\mu F \;\circ\; I + \perp\!\!\!\perp)< \\
= & \quad\{\ \mu F \;=\; G.\mu F + H.\mu F\text{, + abides with composition}\ \} \\
 & (G.\mu F \;+\; \perp\!\!\!\perp)< \\
= & \quad\{\ \text{domains: (58) and (15)}\ \} \\
 & G.\mu F \;+\; \perp\!\!\!\perp
\end{array}
$$

By a completely symmetrical argument we have:

$$\eta> \;=\; H.\mu F \qquad\qquad \text{and} \qquad \eta< \;=\; \perp\!\!\perp + H.\mu F$$

Combining these four domain calculations with the cup and cap abide properties of sum (see theorem 60) and summarising we have established:

Theorem 66 The constructors τ and η are both bijections with the following domain properties:

(a)	$\tau> \;=\; G.\mu F$	(b)	$\tau< \;=\; G.\mu F + \perp\!\!\perp$
(c)	$\eta> \;=\; H.\mu F$	(d)	$\eta< \;=\; \perp\!\!\perp + H.\mu F$
(e)	$\tau< \sqcup \eta< \;=\; \mu F$	(f)	$\tau< \sqcap \eta< \;=\; \perp\!\!\perp$

□

Interpretating these statements in the relational model we have proved that the constructors τ and η establish a (1-1) correspondence between the elements of μF and the elements of the union of $G.\mu F$ and $H.\mu F$ in such a way that elements constructed by τ are distinct from those constructed by η.

Let us now investigate the structure of the catamorphisms of a sum relator. We have:

$$\begin{array}{ll}
 & (\![R]\!) \\
= & \quad \{\ \text{domain trading: theorem 45(b)}\ \} \\
 & (\![R \circ (G.R + H.R)<]\!) \\
= & \quad \{\ +,\ G,\ H \text{ are relators: theorem 35}\ \} \\
 & (\![R \circ G.R< + H.R<]\!) \\
= & \quad \{\ \text{definition } +\text{: (48)}\ \} \\
 & (\![R \circ (\hookrightarrow \circ G.R<) \triangledown (\hookleftarrow \circ H.R<)]\!) \\
= & \quad \{\ \text{spec-junc fusion: theorem 54}\ \} \\
 & (\![(R \circ \hookrightarrow \circ G.R<) \triangledown (R \circ \hookleftarrow \circ H.R<)]\!)
\end{array}$$

This calculation shows that we may assume without loss of generality that for every R there exist specs S and T such that $(\![R]\!) \;=\; (\![S \triangledown T]\!)$. Specifically, $S \;=\; R \circ \hookrightarrow \circ G.R<$ and $T \;=\; R \circ \hookleftarrow \circ H.R<$.

Note that from $(\![R]\!) \;=\; (\![R]\!) \circ \mu F$ and the fact that μF can be expressed as a junc it follows that every catamorphism can also be expressed as a junc. This observation is most useful when combined with the cancellation property of junc (see theorem 59). To see why let us first observe the following instantiation of the junc-cancellation property:

Lemma 67 For $\trianglelefteq \in \{\sqsupseteq, =, \sqsubseteq\}$,

$$X \circ \mu F \trianglelefteq Y \circ \mu F \quad\equiv\quad X \circ \tau \trianglelefteq Y \circ \tau \;\wedge\; X \circ \eta \trianglelefteq Y \circ \eta$$

Proof

$$\begin{array}{ll}
 & X \circ \mu F \;\trianglelefteq\; Y \circ \mu F \\
\equiv & \quad \{\ \text{theorem 65}\ \} \\
 & X \circ \tau \triangledown \eta \;\trianglelefteq\; Y \circ \tau \triangledown \eta \\
\equiv & \quad \{\ \text{spec-junc fusion: theorem 54}\ \} \\
 & (X \circ \tau) \triangledown (X \circ \eta) \;\trianglelefteq\; (Y \circ \tau) \triangledown (Y \circ \eta)
\end{array}$$

$$
\begin{array}{ll}
\equiv & \quad \{ \text{ junc cancellation: theorem 59 } \} \\
& X \circ \tau \trianglelefteq Y \circ \tau \quad \wedge \quad X \circ \eta \trianglelefteq Y \circ \eta
\end{array}
$$

□

Combining lemma 67 with the unique extension property of catamorphisms we derive a characterisation of F-catamorphisms (for sum relators F, of course), namely:

Theorem 68 (UEP for Sum Relators)

$$
\begin{array}{ll}
& X \circ \mu F \;=\; (\!| R \triangledown S |\!) \\
\equiv & \\
& X \circ \tau \;=\; R \circ G.(X \circ \mu F) \quad \wedge \quad X \circ \eta \;=\; S \circ H.(X \circ \mu F)
\end{array}
$$

Proof

$$
\begin{array}{ll}
& X \circ \mu F = (\!| R \triangledown S |\!) \\
\equiv & \quad \{ \text{ catamorphism uep: theorem 42 } \} \\
& X \circ \mu F = R \triangledown S \circ F.(X \circ \mu F) \circ \mu F \\
\equiv & \quad \{ \text{ lemma 67 } \} \\
& \quad X \circ \tau = R \triangledown S \circ F.(X \circ \mu F) \circ \tau \\
& \wedge \quad X \circ \eta = R \triangledown S \circ F.(X \circ \mu F) \circ \eta
\end{array}
$$

Proceeding further with just the first of the conjuncts on the right hand side of the equivalence (the other being completely symmetrical) we have:

$$
\begin{array}{ll}
& R \triangledown S \circ F.(X \circ \mu F) \circ \tau \\
= & \quad \{ \text{ definition of } \tau\text{: theorem 65 } \} \\
& R \triangledown S \circ F.(X \circ \mu F) \circ \mu F \circ \hookrightarrow \\
= & \quad \{ \; \mu F = F.\mu F = \mu F \circ \mu F \; \} \\
& R \triangledown S \circ F.(X \circ \mu F) \circ \hookrightarrow \\
= & \quad \{ \text{ definition of } F\text{: (64), junc-sum fusion: (52) } \} \\
& (R \circ G.(X \circ \mu F)) \triangledown (S \circ H.(X \circ \mu F)) \circ \hookrightarrow \\
= & \quad \{ \text{ junc computation: theorem 55(a) } \} \\
& R \circ G.(X \circ \mu F)
\end{array}
$$

Back-substituting, the desired theorem is obtained.

□

Compared with the general uep property (theorem 42) theorem 68 splits the task of deriving a catamorphism realising a given spec into two separate components, one for each of the constructors. This separation is further reflected in the *computation rules* for τ and η:

Theorem 69 (Computation Rule)

(a) $\quad (\!| R \triangledown S |\!) \circ \tau \;=\; R \circ G.(\!| R \triangledown S |\!)$

(b) $\quad (\!| R \triangledown S |\!) \circ \eta \;=\; S \circ H.(\!| R \triangledown S |\!)$

Proof Instantiate theorem 68 with $X = (\!| R \triangledown S |\!)$ and simplify using the fact that $(\!| R \triangledown S |\!) = (\!| R \triangledown S |\!) \circ \mu F$.

□

Several other properties of sum relators can be derived simply by instantiating the more general properties of catamorphisms listed in section 7, in particular the fusion and monotonicity properties of catamorphisms (theorems 43 and 44). The benefit that is gained is that, in each case, the premise in the theorem can be expressed as a conjunction of two simpler premises, thus decomposing the proof obligations. We postpone performing this exercise, however, until we have added more structure to our class of relators.

9.3 Polymorphically Grounded Relators

A typical characteristic of monotypes occurring in programming problems is that their elements are generated from a base (mono)type by application of one or more operations. For example, the Peano numbers are generated from the set containing just zero by the successor operation. Polymorphic types, such as list or tree, are families of monotypes parameterised by some base (mono)type. We call such types *polymorphically grounded* types (or rather we call their defining relators polymorphically grounded), the word "grounded" referring to the existence of a base monotype. In this section we abstract a definition of "polymorphically grounded" relator. We do this in two steps. First, we abstract what it means for a relator to be grounded. Then, in order to capture the "polymorphic" element, we abstract sufficient conditions for the existence of a "map" operator. We conclude the section with some consequences of the obtained definition.

The mechanism needed to introduce the notion of a ground monotype into our class of relators is straightforward: we consider a sum relator and choose the left component of the sum to be a constant relator, i.e. we consider the case that $G.X = A$ for some monotype A and all specs X, thereby specializing F to the form:

(70) $\quad F.X \;=\; A + H.X$

Using this the constructors are

(71) $\quad \tau \;=\; \mu F \circ \hookrightarrow \;=\; \hookrightarrow \circ A$

(72) $\quad \eta \;=\; \mu F \circ \hookleftarrow \;=\; \hookleftarrow \circ H.\mu F$

The form of the constructors provides some motivation for the chosen restriction on F. Specifically, suppose we interpret monotypes as sets and $f \circ B$, for monotype B and imp f, also as a set, namely the set obtained by applying the function f to the elements of B. Then the set μF is formed by "juncing" two sorts of sets, the set of "ground" elements, i.e. those elements formed by τ, i.e. by applying $\hookrightarrow$ to elements of A, or "non-ground" elements, i.e. those built by η from existing elements of μF. We call relators F satisfying (70) *grounded* relators.

The extra structure introduced into grounded types makes little difference to the computation rule; where it is needed we shall simply instantiate theorem 69(a) with $G.X = A$. The fusion property for ground-relator-catamorphisms is worth stating, however, because we can exploit the extra structure to strengthen the general result.

Theorem 73 (Ground-Relator Fusion) For $\trianglelefteq$ in $\{\sqsubseteq, =, \sqsupseteq\}$,

$$\begin{array}{ll} & U \circ (\!|R \triangledown S|\!) \;\trianglelefteq\; (\!|P \triangledown Q|\!) \\ \Leftarrow & U \circ R \circ A \;\trianglelefteq\; P \circ A \;\;\wedge\;\; U \circ S \circ H.I \;\trianglelefteq\; Q \circ H.U \end{array}$$

□

The added-value of this theorem relative to theorem 43 — apart from the antecedent having been split into two conjuncts — is the introduction of the domain restrictions A and $H.I$ in the first and second conjuncts, respectively, of the antecedent. Note that

$$U \circ R \circ A \ \trianglelefteq\ P \circ A \quad \Leftarrow \quad U \circ R \ \trianglelefteq\ P$$

Thus the first conjunct in the antecedent has been weakened. (That it is a true weakening is easily seen by taking $A = \perp\!\!\perp$.) The second conjunct has been similarly weakened.

Proof Let $\trianglelefteq \in \{\sqsupseteq, =, \sqsubseteq\}$. Then

$$\begin{array}{ll}
 & U \circ (\!|R \triangledown S|\!) \ \trianglelefteq\ (\!|P \triangledown Q|\!) \\
\equiv & \qquad \{ \text{ domain trading: theorem 45(a), since } \quad A{+}H.I \ = \ F.I \\
 & \qquad\quad \text{and junc-sum fusion: theorem 52(a) } \} \\
 & U \circ (\!|(R \circ A) \triangledown (S \circ H.I)|\!) \ \trianglelefteq\ (\!|P \triangledown Q|\!) \\
\Leftarrow & \qquad \{ \text{ catamorphism fusion: theorem 43 } \} \\
 & U \circ (R \circ A) \triangledown (S \circ H.I) \ \trianglelefteq\ P \triangledown Q \circ F.U \\
\equiv & \qquad \{ \text{ spec-junc fusion: theorem 54;} \\
 & \qquad\quad \text{definition of } F\text{: (70),} \\
 & \qquad\quad \text{and + abides with composition: theorem 52(b) } \} \\
 & (U \circ R \circ A) \triangledown (U \circ S \circ H.I) \ \trianglelefteq\ (P \circ A) \triangledown (Q \circ H.U) \\
\equiv & \qquad \{ \text{ junc cancellation: theorem 59(a) } \} \\
 & U \circ R \circ A \ \trianglelefteq\ P \circ A \ \wedge\ U \circ S \circ H.I \ \trianglelefteq\ Q \circ H.U
\end{array}$$

□

We come now to the first of the primitive operators in the Bird-Meertens formalism, namely the map operator. Section 7 provides the appropriate mechanism for introducing such an operator: we must express F in the form $I\otimes$ for some binary relator $\otimes$. This we can do by choosing $A = K.I$ for some relator K and defining the binary relator $\otimes$ by

(74) $$R \otimes S \ = \ K.R + H.S$$

Accordingly we have:

(75) $$F.X \ = \ (I\otimes).X \ = \ K.I + H.X$$

Note that $K.I$ is a monotype so that F is indeed grounded. It is also polymorphic in the sense that we have defined a family of relators, namely the set of relators $(B\otimes)$ for B ranging over all monotypes. More importantly we can instantiate the theorems of section 7 to obtain the sought-after map operator. Specifically, instantiating definition 61 and citing theorem 62, we have:

Theorem 76 (Map) The function ϖ from specs to specs defined by

$$\varpi R \ = \ (\!|K.R + H.I|\!)$$

is a relator.

□

The function ϖ defines a family of monotypes, namely the monotypes ϖB where B ranges over monotypes. In particular, $\varpi I \;=\; \mu F$. For each spec R, the spec ϖR has left domain $\varpi(R<)$ and right domain $\varpi(R>)$. In addition, for monotypes A and B and imps $f \in A \longleftarrow B$, $\varpi f \in \varpi A \longleftarrow \varpi B$. (With the exception of the property $\varpi I \;=\; \mu F$ these properties are valid for all relators, not just map relators.) An instance of such a relator is the List relator which is sometimes denoted by *. In functional programming texts $*f$ is commonly called "map f" (and sometimes written that way too) and denotes a function from lists to lists that "maps" the given function f over the elements of the argument list (i.e. constructs a list of the same length as the argument list whereby the elements are obtained by applying f to each of the elements of the argument list). This then is the origin of the name "map" for ϖ.

We will mostly use another but equivalent definition for map that exploits the particular structure of the relator $\otimes$. That definition is obtained by first instantiating the map fusion theorem (theorem 63) of section 7.

Theorem 77 (Map Fusion)

$$\llparenthesis P \triangledown Q \rrparenthesis \circ \varpi R \;=\; \llparenthesis (P \circ K.R) \triangledown Q \rrparenthesis$$

Proof

$$
\begin{array}{ll}
 & \llparenthesis P \triangledown Q \rrparenthesis \circ \varpi R \\
= & \qquad \{ \text{ map fusion: theorem 63, definition of } \otimes\text{: (74) } \} \\
 & \llparenthesis P \triangledown Q \circ K.R + H.I \rrparenthesis \\
= & \qquad \{ \text{ junc-sum fusion: theorem 52(a) } \} \\
 & \llparenthesis (P \circ K.R) \triangledown Q \circ K.I + H.I \rrparenthesis \\
= & \qquad \{ \text{ domain trading: theorem 45(c), } K.I + H.I \;=\; F.I \ \} \\
 & \llparenthesis (P \circ K.R) \triangledown Q \rrparenthesis
\end{array}
$$

□

Theorem 78 $\quad \varpi R \;=\; \llparenthesis (\tau \circ K.R) \triangledown \eta \rrparenthesis$

Proof

$$
\begin{array}{ll}
 & \varpi R \\
= & \qquad \{ \ \varpi \text{ is a relator } \} \\
 & \varpi I \circ \varpi R \\
= & \qquad \{ \ \varpi I \;=\; \mu F \;=\; \llparenthesis \mu F \rrparenthesis \ \} \\
 & \llparenthesis \mu F \rrparenthesis \circ \varpi R \\
= & \qquad \{ \ \mu F \;=\; \tau \triangledown \eta, \text{ map fusion: theorem 77 } \} \\
 & \llparenthesis (\tau \circ K.R) \triangledown \eta \rrparenthesis
\end{array}
$$

□

The reason why we sometimes prefer this definition is that catamorphisms of the shape $\llparenthesis R \triangledown \eta \rrparenthesis$ enjoy many properties.

Instantiating the computation rule (69) with the revised definition of F — (70) — and the above definition of ϖ we obtain the following computation rules:

$$\varpi R \circ \tau \;=\; \tau \circ K.R \qquad \text{and} \qquad \varpi R \circ \eta \;=\; \eta \circ H.\varpi R$$

These two equations can be recombined into one using theorem 59 viz:

(79) $$\varpi R \circ \tau \triangledown \eta \;=\; (\tau \circ K.R) \triangledown (\eta \circ H.\varpi R)$$

Recalling that

$$\varpi I \;=\; \mu F \;=\; \tau \triangledown \eta \;=\; (\tau \circ K.I) \triangledown (\eta \circ H.\varpi I)$$

(see theorems 46, 65 and equations (71), (72) and (75)) one can view ϖR as a spec which, when applied to an element of μF, applies R to the ground elements but does not destroy the original structure.

9.4 Defining Reduce

The second primitive in the Bird-Meertens formalism is called "reduce" and is denoted by the symbol "/". In the context of our work, reduce is a function from specs to specs. We shall adopt the same symbol but use it as a prefix operator in order to be consistent with our convention of always writing function and argument in that order. Thus we write $/S$ and read "reduce with S" or just "reduce S".

(In choosing to write reduce as a prefix operator we are turning the clock back to Backus' Turing award lecture [6] rather than following the example of Bird and Meertens. In the context of Bird and Meertens' original work reduce was a binary infix operator with argument a pair consisting of a binary operator, say $\oplus$, and a list, say x, thus giving $\oplus/x$. In the course of time it was recognised that calculations and laws could be made more compact by working with the *function* $(x \mapsto \oplus/x)$ rather than the *object* $\oplus/x$. To achieve the compactness the notation $\oplus/$ (or sometimes $(\oplus/)$) was adopted for the function, the process of abstracting one of the arguments of a binary operator being commonly referred to as "sectioning". By this development, presumably, they came to the convention of using "/" as a postfix operator. Since our concern is to profit from what has been learnt rather than repeat the learning process we shall not adopt their notation in its entirety.)

The idea behind reduce is that it should have a complementary behaviour to map. Recall that map, applied to an element of μF, leaves the structure unchanged but applies its argument to the ground elements. Reduce should do the opposite: leave the ground elements unchanged but destroy the structure. Since a catamorphism does both (modifies the ground elements and the structure) we formulate the requirement on reduce as being that every catamorphism is factorisable into a reduce composed with a map. I.e. for all specs R and S,

$$/S \circ \varpi R \;=\; \llparenthesis R \triangledown S \rrparenthesis$$

Let us try to calculate a suitable definition for $/S$.

$$\begin{array}{ll}
 & /S \circ \varpi R \\
= & \qquad \{ \text{ We try to express } /S \text{ as a catamorphism:} \\
 & \qquad\;\; \text{assume } P \text{ and } Q \text{ exist such that: } /S \;=\; \llparenthesis P \triangledown Q \rrparenthesis \;\} \\
 & \llparenthesis P \triangledown Q \rrparenthesis \circ \varpi R \\
= & \qquad \{ \text{ map fusion: theorem 77 } \} \\
 & \llparenthesis (P \circ K.R) \triangledown Q \rrparenthesis
\end{array}$$

Now we cannot choose P and Q (for arbitrary relator K) such that

$$(\!|(P \circ K.R) \ \triangledown \ Q|\!) \ = \ (\!|R \triangledown S|\!)$$

But if we take $P \ = \ I$ and $Q \ = \ S$, i.e. we define the reduce operator by:

(80) $\quad /S \ = \ (\!|K.I \ \triangledown \ S|\!)$

then we have established the following factorisation property:

Lemma 81 $\quad /S \circ \varpi R \ = \ (\!|K.R \ \triangledown \ S|\!)$

□

Some simplification of (80) is possible using domain trading and junc-sum fusion (theorems 45(a) and 52(a)). Specifically, we claim that the term $K.I$ in (80) may be replaced by I (the verification being left to the reader) which leads us to the following definition of reduce:

Definition 82 (Reduce) $\quad /S \ = \ (\!|I \triangledown S|\!)$

□

For $/S$ we have the following computation rules (obtained by instantiating theorem 69 with $G.X = K.I$ for all X):

$$/S \circ \tau \ = \ K.I \qquad \text{and} \qquad /S \circ \eta \ = \ S \circ H./S$$

So one can view $/S$ as a spec which, when applied to an element of μF, strips the ground elements of the constructor τ and replaces the constructor η by S.

9.5 Monadic Relators

As mentioned before, with F having the form given by (75), we cannot factorise every catamorphism into a reduce and a map for arbitrary relator K. For relator K defined by $K.X = X$ — i.e. the identity relator — we can, since

$$
\begin{array}{ll}
 & (\!|R \triangledown S|\!) \\
= & \qquad \{ \ K.R = R \ \} \\
 & (\!|K.R \ \triangledown \ S|\!) \\
= & \qquad \{ \ \text{catamorphism factorisation: theorem 81} \ \} \\
 & /S \circ \varpi R
\end{array}
$$

So we further specialise the binary relator $\otimes$ and the unary relator F by defining

(83) $\quad K.X \ = \ X$

(84) $\quad X \otimes Y \ = \ X + H.Y$

(85) $\quad F.X \ = \ (I\otimes).X \ = \ I + H.X$

for all specs X and Y. Then we have established the all-important:

Theorem 86 (Factorisation) With relator F defined by (84) and (85) we have, for all specs R and S,

$$(\!|R \triangledown S|\!) \quad = \quad /S \circ \varpi R$$

□

The importance of this theorem derives from the fact that it enhances further decomposition of calculations with catamorphisms. Instead of working with the entire catamorphism one works with the components $/S$ and ϖR. Laws are also formulated concerning the individual behaviours of reduce and map as well as their interaction. The advantage is that the laws become extremely compact and thus more manageable, the disadvantage is that there are more of them. Let us illustrate this by considering the computation rules, the unique extension property and the fusion properties of reduce and map.

First, the definitions of the constructors τ and η are specialised accordingly:

(87) $\tau \;=\; \mu F \circ \hookrightarrow \;=\; \hookrightarrow$

(88) $\eta \;=\; \mu F \circ \hookleftarrow \;=\; \hookleftarrow \circ H.\mu F$

Whereas before we had two computation rules, one for each of the constructors, we now have four rules:

Theorem 89 (Computation Rule)

(a) $\varpi R \circ \tau \;=\; \tau \circ R$ (b) $\varpi R \circ \eta \;=\; \eta \circ H.\varpi R$

(c) $/S \circ \tau \;=\; I$ (d) $/S \circ \eta \;=\; S \circ H./S$

□

(Of course these rules can be recombined into two using the factorisation theorem, and whether one chooses to do so is a matter of taste.)

In the case of the unique extension property there is little gain from the use of the factorisation theorem.

Theorem 90 (Unique Extension Property)

$$\begin{array}{ll} & X \circ \mu F \;=\; /S \circ \varpi R \\ \equiv & \\ & X \circ \tau \;=\; R \;\wedge\; X \circ \eta \;=\; S \circ H.(X \circ \mu F) \end{array}$$

□

On the other hand, the fusion law becomes more compact since it suffices to state the law only for a reduce. We call the resulting theorem a "leapfrog" rule because its symbol dynamics is that a reduce "leapfrogs" from one side to the other of a composition of two specs. (The more general fusion law can be recovered by combining the reduce leapfrog theorem with the monotonicity of the relator ϖ.)

Theorem 91 (Reduce Leapfrog) For $\trianglelefteq$ in $\{\sqsupseteq, =, \sqsubseteq\}$,

$$R \circ /S \;\trianglelefteq\; /T \circ \varpi R \quad \Leftarrow \quad R \circ S \circ H.I \;\trianglelefteq\; T \circ H.R$$

Proof

$$\begin{array}{ll}
 & R \circ /S \ \trianglelefteq\ /T \circ \varpi R \\
\equiv & \qquad \{ \text{ definition 82, factorisation: theorem 86 } \} \\
 & R \circ (\![I \triangledown S]\!) \ \trianglelefteq\ (\![R \triangledown T]\!) \\
\Leftarrow & \qquad \{ \text{ ground relator fusion: theorem 73, } \ A = K.I = I \ \} \\
 & R \circ I \circ I \ \trianglelefteq\ R \circ I \quad \wedge \quad R \circ S \circ H.I \ \trianglelefteq\ T \circ H.R \\
\equiv & \qquad \{ \text{ calculus } \} \\
 & R \circ S \circ H.I \ \trianglelefteq\ T \circ H.R
\end{array}$$

□

Because μF is expressible as a catamorphism, it too can be factorised:

Theorem 92 (Identity Rule) $\quad /\eta \circ \varpi\tau \ = \ \varpi I$

Proof

$$\begin{array}{ll}
 & /\eta \circ \varpi\tau \\
= & \qquad \{ \text{ factorisation: theorem 86 } \} \\
 & (\![\tau \triangledown \eta]\!) \\
= & \qquad \{ \text{ constructors: theorem 65 } \} \\
 & (\![\mu F]\!) \\
= & \qquad \{ \text{ identity rules: theorems 46 and 46 } \} \\
 & \varpi I
\end{array}$$

□

Theorem 92 is one of those theorems that, because of their simplicity, are very often overlooked and yet prove to be vital.

A special reduce is $/\eta$ (for list-structures this is the “flattening” catamorphism; it maps a list of lists to a list). For this catamorphism there exist two special leapfrog properties:

Theorem 93 ($/\eta$ Leapfrog)

(a) $\quad /S \circ /\eta \ = \ /S \circ \varpi/S \qquad$ (b) $\quad \varpi R \circ /\eta \ = \ /\eta \circ \varpi\varpi R$

Proof Immediate from the reduce leapfrog rule — theorem 91 — and the two η-computation rules — theorem 89(b) and (d).

□

Corollary 94 The triple (ϖ, τ, α), where $\alpha = /\eta \circ \varpi\varpi I$, is a monad in the following sense:

(a)	ϖ is a relator.	(b)	$\varpi R \circ \tau = \tau \circ R$
(c)	$\varpi R \circ \alpha = \alpha \circ \varpi\varpi R$	(d)	$\alpha \circ \varpi\tau = \varpi I$
(e)	$\alpha \circ \tau = \varpi I$	(f)	$\alpha \circ /\eta = \alpha \circ \varpi\alpha$

Proof Part (a) has already been mentioned. Parts (b) and (e) follow from the computation rule of τ (theorem 89), (c), (d) and (f) follow from theorem 93 together with the identity rule, in the case of (d).

$\Box$

The concept of a monad is highly significant and is given due prominence in the mathematical literature. (See for instance [7, 20]. Note that monads are also called "triples".) In the computing science literature the importance of monads is as yet difficult to assess but appears to be steadily growing, the best known example being lists: a monad is formed by the triple $*$, $[\_]$ and *flatten*, where $*$ denotes the list map operation, $[\_]$ is the function that constructs a singleton list, and *flatten* is the function that "flattens" a list of lists into a single list. See for instance [27] for examples of particular relevance to the design and implementation of functional programming languages.

The existence of a monad structure is the reason why we call the relator of this subsection a "monadic" relator.

9.6 Pointed Relators and Filter

The third, and final, primitive operator in the Bird-Meertens formalism is called "filter" and denoted by $\triangleleft$. The function of $\triangleleft P$ (read "filter with P", or just "filter P") is just to filter out the elements in a given data structure that do not satisfy the predicate P.

There are two obvious requirements on the definition of a filter operation. The first is that $\triangleleft \mathit{true}$ should be the identity function on μF. The second is that $\triangleleft \mathit{false}$ should return an "empty" data-structure. In order to meet the latter requirement we introduce a so-called "unit element" into the definition of H, viz:

(95) $\quad H.X \;=\; 1\!\!1 + J.X$

where J is a relator. Consequently, F is specialised to:

(96) $\quad F.X \;=\; I \;+\; (1\!\!1 + J.X)$

with the two constructors we already have

(97) $\quad \tau \;=\; \mu F \circ \hookrightarrow \;=\; \hookrightarrow$

(98) $\quad \eta \;=\; \mu F \circ \hookleftarrow \;=\; \hookleftarrow \;\circ\; 1\!\!1 + J.\mu F$

and two new ones

(99) $\quad \Box \;=\; \mu F \circ \hookleftarrow \circ \hookrightarrow \;=\; \hookleftarrow \circ \hookrightarrow \circ 1\!\!1$

(100) $\quad +\!\!\!+ \;=\; \mu F \circ \hookleftarrow \circ \hookleftarrow \;=\; \hookleftarrow \circ \hookleftarrow \;\circ\; J.\mu F$

Note that

(101) $\quad \eta \;=\; \Box \triangledown +\!\!\!+$

Because this relator has a disjoint unit in its ground as well, we call these relators "pointed relators". Again we want to point out that because this relator F is just an instance of the previous one, the definition of map and reduce stay the same and all the theorems stated so far remain valid. For our immediate purposes we only need to update the computation rule:

Theorem 102 (Computation Rule) In addition to the computation rules given in theorem 89 we have:

(a) $\varpi R \circ \Box \;=\; \Box$
(b) $\varpi R \circ +\!\!\!+ \;=\; +\!\!\!+ \circ J.\varpi R$
(c) $/(S \triangledown T) \circ \Box \;=\; S \circ \mathbb{1}$
(d) $/(S \triangledown T) \circ +\!\!\!+ \;=\; T \circ J./(S \triangledown T)$

Proof There are two pairs of computation rules given in the theorem but by using junc cancellation (theorem 59(a)) we can derive the elements of each pair simultaneously. We illustrate the method on the second pair:

$$
\begin{array}{ll}
 & (/(S \triangledown T) \circ \Box) \;\triangledown\; (/(S \triangledown T) \circ +\!\!\!+) \\
= & \qquad \{ \text{ spec-junc fusion: theorem 54(a) } \} \\
 & /(S \triangledown T) \circ \Box \triangledown +\!\!\!+ \\
= & \qquad \{ \text{ (101) } \} \\
 & /(S \triangledown T) \circ \eta \\
= & \qquad \{ \text{ computation rule: theorem 89(d) } \} \\
 & S \triangledown T \;\circ\; \mathbb{1} + J./(S \triangledown T) \\
= & \qquad \{ \text{ junc-sum fusion: theorem 52(a) } \} \\
 & (S \circ \mathbb{1}) \;\triangledown\; (T \circ J./(S \triangledown T))
\end{array}
$$

We have thus proved the equality of two juncs. Rules (c) and (d) now follow by the junc cancellation theorem. The first pair is derived similarly.
□

The definition of filter is borrowed directly from the work of Meertens [23] and Bird [12]:

Definition 103 (Filter) For all specs P,

$$\triangleleft P \;=\; /\eta \circ \varpi(\tau \lhd P \rhd (\Box \circ \top\!\!\top))$$

□

Note that from the fact that τ and $\Box \circ \top\!\!\top$ are imps and the fact that conditionals, junc and catamorphism respect imps it follows that $\triangleleft P$ is an imp.

In this section we explore several algebraic properties of the filter operation. The properties that we seek are motivated by the relationship between the Bird-Meertens formalism and the quantifier calculus often used in the derivation of imperative programs (e.g. [1, 16]).

By design $\triangleleft true$ is the identity function on specs of the correct type:

Theorem 104 $\triangleleft true \;=\; \varpi I$

Proof

$$
\begin{array}{ll}
 & \triangleleft true \\
= & \qquad \{ \text{ definition 103 } \} \\
 & /\eta \circ \varpi(\tau \lhd true \rhd (\Box \circ \top\!\!\top))
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \text{ conditionals: theorem 33(a) } \} \\
& /\eta \circ \varpi\tau \\
= & \{ \text{ identity rule: theorem 92 } \} \\
& \varpi I
\end{array}
$$

□

Now we consider whether two filters can be fused into one. Since $\triangleleft P$ is a catamorphism of the form $/\eta \circ \varpi p$ where $p = \tau \lhd P \rhd (\Box \circ \top\!\top)$ it pays to begin by exploring whether a map can be fused with a filter. Indeed it can.

Lemma 105

(a) $\varpi R \circ \triangleleft P = /\eta \circ \varpi((\tau \circ R) \lhd P \rhd (\Box \circ \top\!\top))$
(b) $/\eta \circ \varpi R \circ \triangleleft P = /\eta \circ \varpi(R \lhd P \rhd (\Box \circ \top\!\top))$

Proof For brevity let p denote $\tau \lhd P \rhd (\Box \circ \top\!\top)$. Then we prove part (a) as follows:

$$
\begin{array}{ll}
& \varpi R \circ \triangleleft P \\
= & \{ \text{ definition 103 } \} \\
& \varpi R \circ /\eta \circ \varpi p \\
= & \{ \ /\eta \text{ leapfrog: theorem 93(b) } \} \\
& /\eta \circ \varpi\varpi R \circ \varpi p \\
= & \{ \ \varpi \text{ is a relator, definition of } p \ \} \\
& /\eta \circ \varpi(\varpi R \circ \tau \lhd P \rhd (\Box \circ \top\!\top)) \\
= & \{ \text{ conditionals: theorem 33(n) } \} \\
& /\eta \circ \varpi((\varpi R \circ \tau) \lhd P \rhd (\varpi R \circ \Box \circ \top\!\top)) \\
= & \{ \text{ computation rule: theorem 102(a) } \} \\
& /\eta \circ \varpi((\tau \circ R) \lhd P \rhd (\Box \circ \top\!\top))
\end{array}
$$

Part (b) is derived from (a) using the leapfrog rule, theorem 93(a), followed by theorem 33(n) and the computation rule 102(c).

□

A direct consequence of lemma 105 is:

Theorem 106 ($\triangleleft$ distribution) $\triangleleft P \circ \triangleleft Q = \triangleleft (P \wedge Q)$

Proof

$$
\begin{array}{ll}
& \triangleleft P \circ \triangleleft Q \\
= & \{ \text{ definition 103, lemma 105(b) } \} \\
& /\eta \circ \varpi((\tau \lhd P \rhd (\Box \circ \top\!\top)) \lhd Q \rhd (\Box \circ \top\!\top)) \\
= & \{ \text{ conditionals: theorem 33(f) } \} \\
& /\eta \circ \varpi(\tau \lhd (P \wedge Q) \rhd (\Box \circ \top\!\top)) \\
= & \{ \text{ definition 103 } \} \\
& \triangleleft (P \wedge Q)
\end{array}
$$

□

We conclude with yet another translation rule.

Theorem 107 (Filter Translation) For all imps f

$$\triangleleft P \circ \varpi f \;=\; \varpi f \circ \triangleleft(P \circ f) \circ \varpi f{>}$$

Proof

$$\begin{array}{ll}
 & \varpi f \circ \triangleleft(P \circ f) \circ \varpi f{>} \\
= & \qquad \{ \text{ lemma 105(a) } \} \\
 & /\eta \circ \varpi((\tau \circ f) \lhd (P \circ f) \rhd (\Box \circ \top\!\!\top)) \circ \varpi f{>} \\
= & \qquad \{ \text{ relator.}\varpi \text{ } \} \\
 & /\eta \circ \varpi((\tau \circ f) \lhd (P \circ f) \rhd (\Box \circ \top\!\!\top) \circ f{>}) \\
= & \qquad \{ \text{ imp.}(f{>})\text{, conditionals: theorem 33(o) } \} \\
 & /\eta \circ \varpi((\tau \circ f \circ f{>}) \lhd (P \circ f \circ f{>}) \rhd (\Box \circ \top\!\!\top \circ f{>})) \\
= & \qquad \{ \text{ domains: (14) and } \top\!\!\top \circ f{>} \;=\; \top\!\!\top \circ f : (11) \text{ } \} \\
 & /\eta \circ \varpi((\tau \circ f) \lhd (P \circ f) \rhd (\Box \circ \top\!\!\top \circ f)) \\
= & \qquad \{ \bullet \text{ imp.}f\text{, conditionals: theorem 33(o) } \} \\
 & /\eta \circ \varpi(\tau \lhd P \rhd (\Box \circ \top\!\!\top) \circ f) \\
= & \qquad \{ \text{ relator.}\varpi \text{ } \} \\
 & /\eta \circ \varpi(\tau \lhd P \rhd (\Box \circ \top\!\!\top)) \circ \varpi f \\
= & \qquad \{ \text{ definition 103 } \} \\
 & \triangleleft P \circ \varpi f
\end{array}$$

$\Box$

Theorem 107 can also be strengthened in the same way that theorem 33(o) was strengthened to theorem 33(p).

The syntactic resemblance of theorems 25 and 107 should not go unnoticed. After some thought the resemblance is not surprising: $P{>}$ is a sort of filter but on elements of some base set, $\triangleleft P$ is the same filter but "lifted" to elements of ϖI.

References

1. R.C. Backhouse. *Program Construction and Verification.* Prentice-Hall International, 1986.
2. R.C. Backhouse. An exploration of the Bird-Meertens formalism. Technical Report CS8810, Department of Mathematics and Computing Science, University of Groningen, 1988.
3. R.C. Backhouse. Calculating the Warshall/Floyd path algorithm. Eindhoven University of Technology, Department of Computing Science, Computer Science Note No. 92/09, May 1992.
4. R.C. Backhouse, P. de Bruin, P. Hoogendijk, G. Malcolm, T.S. Voermans, and J. van der Woude. Polynomial relators. In M. Nivat, C.S. Rattray, T. Rus, and G. Scollo, editors, *Proceedings of the 2nd Conference on Algebraic Methodology and Software Technology, AMAST'91.* Springer-Verlag, Workshops in Computing, 1992.
5. R.C. Backhouse, P. de Bruin, G. Malcolm, T.S. Voermans, and J. van der Woude. Relational catamorphisms. In Möller B., editor, *Proceedings of the IFIP TC2/WG2.1 Working Conference on Constructing Programs*, pages 287–318. Elsevier Science Publishers B.V., 1991.
6. J. Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, August 1978.

7. M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer-Verlag, 1985.
8. R.S. Bird. The promotion and accumulation strategies in transformational programming. *ACM. Transactions on Programming Languages and Systems*, 6(4):487–504, 1984.
9. R.S. Bird. Transformational programming and the paragraph problem. *Science of Computing Programming*, 6:159–189, 1986.
10. R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design.* Springer-Verlag, 1987. NATO ASI Series, vol. F36.
11. R.S. Bird. A calculus of functions for program derivation. Technical report, Programming Research Group, Oxford University, 11, Keble Road, Oxford, OX1 3QD, U.K., 1988.
12. R.S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science*, pages 151–216. Springer-Verlag, 1989. NATO ASI Series, vol. F55.
13. R.S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. Technical report, Programming Research Group, Oxford University, 11, Keble Road, Oxford, OX1 3QD, U.K., 1988.
14. R.S. Bird and L. Meertens. Two exercises found in a book on algorithmics. In L.G.L.T. Meertens, editor, *Program Specification and Transformations*, pages 451–457. Elsevier Science Publishers B.V., North Holland, 1987.
15. E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
16. E.W. Dijkstra and W.H.J. Feijen. *Een Methode van Programmeren.* Academic Service, Den Haag, 1984. Also available as *A Method of Programming*, Addison-Wesley, Reading, Mass., 1988.
17. C.A.R. Hoare *et al.* Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987. Corrigenda in *30*, 9, p. 770.
18. P.F. Hoogendijk. (Relational) Programming laws in the Boom hierarchy of types. To appear: Conference Proceedings, Mathematics of Program Construction, Oxford UK, June, 1992.
19. G. Hutton and E. Voermans. Making functionality more general. In *Functional Programming, Glasgow 1991*, Workshops in computing. Springer-Verlag, 1991. (To appear).
20. J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Studies in Advanced Mathematics.* Cambridge University Press, 1986.
21. G. Malcolm. Homomorphisms and promotability. In J.L.A. van de Snepscheut, editor, *Conference on the Mathematics of Program Construction*, pages 335–347. Springer-Verlag LNCS 375, 1989.
22. E.G. Manes and M.A. Arbib. *Algebraic Approaches to Program Semantics.* Texts and Monographs in Computer Science. Springer-Verlag, Berlin, 1986.
23. L. Meertens. Algorithmics – towards programming as a mathematical activity. In *Proceedings of the CWI Symposium on Mathematics and Computer Science*, pages 289–334. North-Holland, 1986.
24. F.J. Rietman. A note on extensionality. In J. van Leeuwen, editor, *Proceedings Computer Science in the Netherlands 91*, pages 468–483, 1991.
25. G. Schmidt and T. Ströhlein. *Relationen und Grafen.* Springer-Verlag, 1988.
26. E. Voermans. Pers as types, inductive types and types with laws. In *PHOENIX Seminar and Workshop on Declarative Programming, Sasbachwalden*, Workshops in Computing. Springer-Verlag, 1991. (To appear).
27. P. Wadler. Comprehending monads. In *ACM Conference on Lisp and Functional Programming*, June 1990.

This article was processed using the LaTeX macro package with LMAMULT style