(Relational) Programming Laws in the Boom Hierarchy of Types

Paul F. Hoogendijk*

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

Abstract. In this paper we demonstrate that the basic rules and calculational techniques used in two extensively documented program derivation methods can be expressed, and, indeed, can be generalised within a relational theory of datatypes. The two methods to which we refer are the so-called "Bird-Meertens formalism" (see [22]) and the "Dijkstra-Feijen calculus" (see [15]).

The current paper forms an abridged, though representative, version of a complete account of the algebraic properties of the Boom hierarchy of types [19, 18]. Missing is an account of extensionality and the so-called cross-product.

1 Introduction

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 [22] and has been further refined and applied in a number of papers by Bird and Meertens [8, 9, 11, 12, 13]. Its beauty derives from the small scale of the theory itself compared with the large scale of applications.

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" in line with Backus [7]; Iverson [20] used the name "reduce". 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" [2] after H.J. Boom to whom Meertens attributes the concept.

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

⋆ The investigations were (partly) supported by the Foundation for Computer Science in the Netherlands (SION) with financial support from the Netherlands Organization for Scientific Research (NWO)

2 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.

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 [3] we exploit the power of abstraction afforded by the notion of a so-called relator (a relator is a generalization of a functor) to add several more levels to the Boom hierarchy each of which is "poorer" than those considered by Meertens. 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. 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. In the present paper, we start with (an instantiation of) pointed relators and give the definition of map, reduce and filter. For further discussion we refer to the paper [3].

The unconventional nature (and perhaps also the conciseness) of the notations used in the Bird-Meertens formalism makes the formalism difficult to comprehend for many groups. The program calculations carried out within the formalism are, however, strongly related to calculations within other systems. In particular there is a strong link between a certain combination of the three basic operators of the formalism and the quantifier expressions used for many years in the Eindhoven school of program development, this link being expressed via a correspondence between the basic laws of the two systems. For the benefit of those familiar with the Eindhoven calculus we use the opportunity to point out elements of this correspondence. What emerges is that there are typically more laws in the Bird-Meertens formalism than the quantifier calculus but the Bird-Meertens formalism exhibits a much better developed separation of concerns.

The theorems presented in the current paper are more general than those in the publications of Bird and Meertens since their work is restricted to total functions. (Meertens [22] does discuss the issue of indeterminacy but this part of his paper — we regret to have to say — is in our view the least satisfactory.) 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.

In [1] a rigorous discussion of the "Bird-Meertens formalism" can be found. This report was the starting point for the present paper. We start with a brief introduction to a relational calculus of datatypes as described in [4, 6]. Thereafter, we define the so-called binary structures; also we define the map, reduce and filter operators in our system. Before we start with the original Boom hierarchy where laws play an important rôle, we first define what it means for a relation to be associative, symmetric or idempotent; also we give a definition for sectioning and units. Next we define the original Boom-hierarchy in our system. Many properties from E. Voermans' paper [24] will be used. That paper describes how laws can be incorporated into the relational calculus. Finally, we relate the formalism to the quantifier calculus. We prove rules like range translation, trading, range splitting, etc., within our formalism.

1.1 Plat Calculus

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, [23].

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 $(\mathcal{A}, \sqcap, \sqcup, \neg, \top\!\!\top, \perp\!\!\perp)$ where "$\sqcap$" and "$\sqcup$" are associative and idempotent binary infix operators with unit elements "$\top\!\!\top$" and "$\perp\!\!\perp$", respectively, and "$\neg$" is the unary prefix operator denoting complement (or negation). We assume familiarity with the standard definition of a lattice given above. We call such a structure a *plat*, the "p" standing for power set and "lat" standing for lattice.

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 $V, W \subseteq \mathcal{A}$, $(\sqcup V) \circ (\sqcup W) = \sqcup(R, S \ : \ R \in V \wedge S \in W \ : \ R \circ S)$

The third layer is the reverse structure: $(\mathcal{A}, {}^{\cup})$ where "${}^{\cup}$" is a unary postfix operator such that it is its own inverse. The interface with the first layer is that "${}^{\cup}$" is an isomorphism of plats. I.e. for all $R, S \in \mathcal{A}$, $R^{\cup} \sqsupseteq S \ \equiv \ R \sqsupseteq S^{\cup}$. The interface with the second layer is that "${}^{\cup}$" is a contravariant monoid isomorphism: $(R \circ S)^{\cup} \ = \ S^{\cup} \circ R^{\cup}$.

A model for this axiom system is the set of binary relations over some universe, with $\sqcup$, $\sqcap$ and $\sqsupseteq$ interpreted as set union, set intersection and set containment, respectively.

1.2 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.

1.3 The Middle Exchange Rule and the Cone Rule

To the above axioms we now add an axiom relating all three layers:
Middle Exchange Rule: $X \sqsupseteq R \circ Y \circ S \;\equiv\; \neg Y \sqsupseteq R^{\cup} \circ \neg X \circ S^{\cup}$
Our last axiom, which is sometimes referred to as "Tarski's Rule", we call the
Cone Rule: $\top\!\!\top \circ R \circ \top\!\!\top = \top\!\!\top \;\equiv\; R \neq \bot\!\!\bot$

1.4 Imps, totality, co-imps and surjectivity

The notions of functionality, totality, injectivity and surjectivity can be defined using the operations introduced above. To avoid confusion between object language and metalanguage we use the term *imp* instead of *function*:

Definition 1

(a) R is an *imp* iff $R \circ R^{\cup} \sqsubseteq I$

(b) R is *total* iff $R^{\cup} \circ R \sqsupseteq I$

(c) R is a *co-imp* iff $R^{\cup}$ is an imp

(d) R is *surjective* iff $R^{\cup}$ is total

□

The term *imp* is derived from the word *imp*lementation. By interpretation, the definition of imps says that R is zero- or single-valued; the definition of totality means that R always returns some result.

1.5 Domains

We say that spec A is a *monotype* iff $I \sqsupseteq A$. Monotypes may be interpreted as an implementation of sets: element x is contained in the set corresponding to monotype A iff $x\langle A\rangle x$.

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, and defined by:

Definition 2 *domains*

(a) $R< \;=\; R \circ R^{\cup} \sqcap I$

(b) $R> \;=\; R^{\cup} \circ R \sqcap I$

□

Note that domains are monotypes. Moreover, $R<$ is the smallest monotype satisfying the following equation:

(3) $X :: \;\; R = X \circ R$

One of the main considerations of the development of the theory described in [6] was that the notion of domain is a part of the system itself: it is not something defined outside the system. This means that we can use type-information in our calculations in a very natural way. So, type considerations are a part of a particular law itself rather than being expressed in some overall context within which the law is applicable.

1.6 Polynomial Relators

In categorical approaches to type theory a parallel is drawn between the notion of type constructor and the categorical notion of "functor", thereby emphasising that a type constructor is not just a function from types to types but also comes equipped with a function that maps arrows to arrows. In [4, 6] an extension to the notion of functor is given, the so-called "relator":

Definition 4 A *relator* is a function, F, from specs to specs such that

(a) $I \sqsupseteq F.I$ (b) $R \sqsupseteq S \Rightarrow F.R \sqsupseteq F.S$
(c) $F.(R \circ S) = F.R \circ F.S$ (d) $F.(R^{\cup}) = (F.R)^{\cup}$
□

One of the main properties of a relator is that it commutes with the domain operators, $F.(R>) = (F.R)>$. Another is that relators respect imps.

1.7 Disjoint Sum and Cartesian Product

We begin by postulating the existence of four specs: the two projections $\ll$ (pronounced "project left") and $\gg$ (pronounced "project right") for cartesian product and the two injections $\hookrightarrow$ (pronounced "inject left",) and $\hookleftarrow$ (pronounced "inject right") for disjoint sum. Further, experience leads us to introduce four binary operators on specs: for cartesian product $\vartriangle$ (pronounced "split", commonly the notation $\langle \_,\_ \rangle$ is used) and $\times$ (pronounced "times"), and for disjoint sum $\triangledown$ (pronounced "junc", commonly the notation $[\_,\_]$ is used) and $+$ (pronounced "plus"), defined in terms of the projection and injection specs as follows:

(5) $P \vartriangle Q = (\ll^{\cup} \circ P) \sqcap (\gg^{\cup} \circ Q)$
(6) $P \triangledown Q = (P \circ \hookrightarrow^{\cup}) \sqcup (Q \circ \hookleftarrow^{\cup})$
(7) $P \times Q = (P \circ \ll) \vartriangle (Q \circ \gg)$
(8) $P+Q = (\hookrightarrow \circ P) \triangledown (\hookleftarrow \circ Q)$

The relational model that we envisage assumes that the universe is a term algebra formed by closing some base set under three operators: the binary operator mapping the pair of terms x, y to the term (x, y), and two unary operators $\hookrightarrow$ and $\hookleftarrow$ mapping the term x to the terms $\hookrightarrow.x$ and $\hookleftarrow.x$, respectively. The interpretation of $\ll$ and $\gg$ is that they project a pair onto its left and right components. The operators defined above have a higher precedence than composition "$\circ$". In [4, 6] the axioms are given for the projections and injections such that cartesian product and disjoint sum have the desired properties. One of these properties is that cartesian product and disjoint sum are (binary) relators. Furthermore, $\ll$, $\gg$ are imps and $\hookrightarrow$, $\hookleftarrow$ are imps and co-imps. Also, we have the following properties:

(9) $P \vartriangle Q \circ R \sqsubseteq (P \circ R) \vartriangle (Q \circ R)$
(10) $P \vartriangle Q \circ f = (P \circ f) \vartriangle (Q \circ f)$, for imp f
(11) $R \circ P \triangledown Q = (R \circ P) \triangledown (R \circ Q)$

Property (10) is called split-imp fusion and property (11) is called spec-junc fusion.

1.8 Unit

The last axiom we add is the existence of a unit type. The unit type (denoted $1\!\!1$), when viewed as a set of pairs, consists of at most one pair, the two components of which are identical:

Unit: $\perp\!\!\perp \neq 1\!\!1 \;\wedge\; I \;\sqsupseteq\; 1\!\!1 \circ \top\!\!\top \circ 1\!\!1$

One of the main properties of $1\!\!1$ is that it is a monotype and that it is an atom.

2 Binary Structures

Throughout this paper we consider the following so-called "pointed relator":

(12) $F.X \;=\; I + (1\!\!1 + X \times X)$

For this relator F we define μF to be the least solution of $X :: X = F.X$. One of the main properties of μF is that μF is a monotype (see [5, 6]). One can view μF as the recursively defined type corresponding to the relator F. For μF we have the following four constructors:

(13) $\tau \;=\; \mu F \circ \hookrightarrow \;=\; \hookrightarrow$
(14) $\eta \;=\; \mu F \circ \hookleftarrow \;=\; \hookleftarrow \circ\; 1\!\!1 + \mu F \times \mu F$
(15) $\Box = \mu F \circ \hookleftarrow \circ \hookrightarrow \;=\; \hookleftarrow \circ \hookrightarrow \circ 1\!\!1$
(16) $+\!\!+ \;=\; \mu F \circ \hookleftarrow \circ \hookleftarrow \;=\; \hookleftarrow \circ \hookleftarrow \circ\; \mu F \times \mu F$

Informally stated, τ is the singleton constructor: from an element of the universe I it constructs a singleton containing that element. $\Box$ is the unit constructor and $+\!\!+$ is the join operator: from two elements of μF it constructs their join. Note that

(17) $\eta = \Box \triangledown +\!\!+$

So, η is just the combination of two other constructors. Elements of μF we call bins (short for binary structures); bins correspond to the D-structures which Meertens describes as "about the poorest possible algebra".

Corresponding with μF we can define the so-called catamorphism operator denoted by $(\!|\_|\!)$. Catamorphism $(\!|R|\!)$ is defined to be the least solution of

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

Catamorphisms can be viewed as a recursively defined specs which follow the same recursion pattern as the elements of μF. For a further account of catamorphisms see [5, 6].

We may assume without loss of generality that every catamorphism corresponding with the relator F can be written as $(\!|R \triangledown (S \triangledown \otimes)|\!)$ with $\otimes> \;\sqsubseteq\; I \times I$. We call specs with right domains contained in $I \times I$ *binary specs*. From now on $\otimes$ denotes a binary spec.

For the constructors defined above we have the following computation rules:

Theorem 18 (Computation Rule)

(a) $\llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis \circ \tau \;=\; R$
(b) $\llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis \circ \Box \;=\; S \circ \mathbb{1}$
(c) $\llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis \circ \mathbin{+\!\!+} \;=\; \otimes \circ \llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis \times \llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis$
$\Box$

For later use we define the following shorthand:

Definition 19 (Empty Tree)

$$\varepsilon \;=\; \Box \circ \top\!\!\top$$

$\Box$

ε can be viewed as the constant function always returning $\Box <$, i.e. the unit element. Functions like ε are so-called points. The definition of a point is:

Definition 20 (Point) Spec x is a *point* iff

$$x \text{ is an imp} \quad \wedge \quad x \;=\; x \circ \top\!\!\top$$
$\Box$

As mentioned above, points are constant functions: applied to an arbitrary element they always return the same element.

We also need the following shorthands:

(21) $R \otimes S \;=\; \otimes \circ R \vartriangle S$
(22) $R \underline{\otimes} S \;=\; (R \circ \ll) \otimes (S \circ \gg) \;=\; \otimes \circ R \times S$

For $R \otimes S$ we have by split-imp fusion (10):

(23) $R \otimes S \circ f \;=\; (R \circ f) \otimes (S \circ f)$, for imp f

Note that the computation rule for $\mathbin{+\!\!+}$ can now be rewritten as:

(24) $\llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis \circ \mathbin{+\!\!+} \;=\; \llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis \underline{\otimes} \llparenthesis R \triangledown (S \triangledown \otimes) \rrparenthesis$

2.1 Map and Reduce

Because relator F is of the shape $I\oplus$, i.e. $F.X \;=\; I \oplus X$ where $\oplus$ is the binary relator defined by $R \oplus S \;=\; R + (\mathbb{1} + S \times S)$, we can define a map operator according to section 5 of [5] or section 7 of [6]. Instantiating the definition given there with our relator, we get:

Theorem 25 (Map) The function $*$ from specs to specs defined by

$$*R \;=\; \llparenthesis R + (\mathbb{1} + I \times I) \rrparenthesis$$

is a relator.
$\Box$

The function $*$ defines a family of monotypes, namely the monotypes $*B$ where B ranges over monotypes. In particular, $*I = \mu F$. For each spec R, the spec $*R$ has left domain $*(R<)$ and right domain $*(R>)$. In addition, for monotypes A and B and imps $f \in A \longleftarrow B$, $*f \in *A \longleftarrow *B$. An instance of such a relator is the List relator. 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, where 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 $*$.

We will mostly use a different but equivalent definition for map that exploits the particular structure of the relator $\oplus$. That definition is obtained by using the following theorem:

Theorem 26 (Map Fusion)

$$(\!|P \triangledown Q|\!) \circ *R \;=\; (\!|(P \circ R) \triangledown Q|\!)$$

$\Box$

Theorem 27 (Map – Alternative Definition)

$$*R \;=\; (\!|(\tau \circ R) \triangledown \eta|\!)$$

$\Box$

The reason why we sometimes prefer this definition is that catamorphisms of the shape $(\!|R \triangledown \eta|\!)$ enjoy many properties.

Instantiating the computation rule (18) with the above definition of $*$ we obtain the following computation rules:

$$\begin{aligned} *R \circ \tau &= \tau \circ R \\ *R \circ \Box &= \Box \\ *R \circ +\!\!+ &= +\!\!+ \circ *R \times *R \end{aligned}$$

One can view $*R$ as a spec which, when applied to an element of μF, applies R to the ground elements but does not destroy the original structure.

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 [7] 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,

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

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

$$\begin{array}{ll}
 & /S \circ *R \\
= & \qquad \{ \text{ We try to express } /S \text{ as a catamorphism} \\
 & \qquad\quad \bullet\ /S = (\!|P \triangledown Q|\!) \ \} \\
 & (\!|P \triangledown Q|\!) \circ *R \\
= & \qquad \{ \text{ map fusion: theorem 26 } \} \\
 & (\!|(P \circ R) \triangledown Q|\!) \\
= & \qquad \{ \text{ choose } P = I \text{ and } Q = S \ \} \\
 & (\!|R \triangledown S|\!)
\end{array}$$

Thus if we define the reduce operator by:

(28) $/S = (\!|I \triangledown S|\!)$

then we have established the following factorisation property:

Lemma 29

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

□

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 well-known leapfrog properties:

Theorem 30 ($/\eta$ Leapfrog)

(a) $/S \circ /\eta = /S \circ */S$
(b) $*R \circ /\eta = /\eta \circ **R$
□

2.2 Conditionals

Conditionals (**if-then-else** statements) are, of course, a well-established feature of programming languages. 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].

In order to be able to define conditionals we need to have the complement of monotypes. We achieve this in a slightly roundabout way. That is to say, we consider the right domains of so-called right conditions. For right conditions we have the following definition:

Definition 31 (Right Condition) We call spec p a right condition if

$$p \;=\; \top\!\!\top \circ p$$

□

(Of course we have a dual definition for left conditions.) We adopt the convention that lower case letters p, q and r denote right conditions.

In the relational model right conditions may be used to represent boolean tests: the predicate b may be represented by the right condition p where for all elements x and y

$$x\langle p\rangle y \;\;\equiv\;\; b.y$$

Now the complement of the monotype $p>$ is just $(\neg p)>$, i.e. we have the properties $p> \sqcup (\neg p)> \;=\; I$ and $p> \sqcap (\neg p)> \;=\; \perp\!\!\perp$. Right conditions are closed under union, intersection, negation, and right composition. In order to give the properties of conditionals a more familiar appearance we shall write $p \wedge q$ instead of $p \sqcap q$, and $p \vee q$ instead of $p \sqcup q$. Other familiar boolean operators can then be defined on right conditions, such as $p \Rightarrow q \;\;=\;\; \neg p \vee q$. We shall also sometimes write *true* instead of $\top\!\!\top$ and *false* instead of $\perp\!\!\perp$. We define:

Definition 32 (Conditional) For all right conditions p we define the binary operator $\lhd p \rhd$ by:

$$R \lhd p \rhd S \;\;=\;\; R \circ p> \;\sqcup\; S \circ (\neg p)>$$

□

Notes: In [16] Hoare defines conditionals, although for propositions, as follows,

$$P \lhd Q \rhd R \;=\; (P \sqcap Q) \;\sqcup\; (R \sqcap \neg Q)$$

Note that our definition corresponds to Hoare's definition because

$$R \circ p> \;\sqcup\; S \circ (\neg p)> \;=\; (R \sqcap p) \;\sqcup\; (S \sqcap \neg p)$$

In [3] an alternative definition of conditionals is proposed that eliminates the need to restrict p to a right condition. Moreover, a more comprehensive account of their properties is included.

End of Notes.

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.

For conditionals we have the commonly known properties which one can find for instance in [17]. The subset of those properties that we use in the present paper is:

Theorem 33 For all specs R, S, T, conditions p and q, and imp f:

(a) $R \lhd \mathit{true} \rhd S \;=\; R$
(b) $R \lhd \mathit{false} \rhd S \;=\; S$
(c) $R \lhd p \rhd R \;=\; R$
(d) $R \lhd \neg p \rhd S \;=\; S \lhd p \rhd R$
(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) $(R \lhd p \rhd S) \lhd q \rhd T \;=\; (R \lhd q \rhd T) \lhd p \rhd (S \lhd q \rhd T)$
(i) $(R \lhd p \rhd S) \vartriangle T \;=\; (R \vartriangle T) \lhd p \rhd (S \vartriangle T)$
(j) $T \circ R \lhd p \rhd S \;=\; (T \circ R) \lhd p \rhd (T \circ S)$
(k) $R \lhd p \rhd S \circ f \;=\; (R \circ f) \lhd (p \circ f) \rhd (S \circ f)$
□

Note that for property (k) the condition that f is an imp is necessary. Property (k) is called the range translation rule for conditionals.

2.3 Definition of Filters

The definition of filter is borrowed directly from the work of Meertens [22] and Bird [10]:

Definition 34 (Filter) For right-condition p,

$$\triangleleft p \;=\; /\eta \circ *(\tau \lhd p \rhd \varepsilon)$$

□

Note that from the fact that τ and ε are imps and the fact that conditionals, junc and catamorphisms respect imps it follows that $\triangleleft p$ is an imp.

In this section we explore some algebraic properties of the filter operation. The properties that we seek are motivated by the relationship between the Bird-Meertens formalism and the so-called quantifier calculus, which relationship will be clarified in section 5.

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

Theorem 35

$$\triangleleft \mathit{true} \;=\; *I$$

□

Now we consider whether two filters can be fused into one. Since $\triangleleft p$ is a catamorphism of the form $/\eta \circ *\overline{p}$ where $\overline{p} \;=\; \tau \lhd p \rhd \varepsilon$ it pays to begin by exploring whether a map can be fused with a filter. Indeed it can.

Lemma 36

(a) $*R \circ \triangleleft p \;=\; /\eta \circ *((\tau \circ R) \lhd p \rhd \varepsilon)$
(b) $/\eta \circ *R \circ \triangleleft p \;=\; /\eta \circ *(R \lhd p \rhd \varepsilon)$
□

A direct consequence of lemma 36b is:

Theorem 37 ($\triangleleft$ distribution)

$$\triangleleft p \;\circ\; \triangleleft q \quad = \quad \triangleleft (p \wedge q)$$

□

Yet another fusion property for filters is

Theorem 38 (Filter Range Translation) For all imps f,

$$\triangleleft p \;\circ\; * f \quad = \quad * f \;\circ\; \triangleleft (p \circ f) \;\circ\; * f >$$

□

3 Sectioning and Laws

In this section we give definitions for left- and right- sectioning, symmetry, associativity, idempotency and left- and right units. For each of these notions we derive a number of properties.

3.1 Sectioning

One important notion we know for binary functions is sectioning, i.e. we can construct a unary function from a binary function, by fixing one of the arguments. For instance (1+) is defined to be the function such that

$$\forall (x :: \; (1+).x = 1+x)$$

In the relational setting, the fixed argument does not necessarily have to be a point; we can generalize it to a left condition because left conditions can be viewed as the relational generalization of constants. We define left-sectioning by

Definition 39 (Left-Sectioning) For p a left condition,

$$p\otimes \quad = \quad p \otimes I$$

□

For the spec $p\otimes$ we then have the desired property:

Theorem 40 (Left-Sectioning)

$$p\otimes \;\circ\; R \;=\; p \otimes R$$

□

Note that by taking $R := I$ it follows that $p\otimes$ is the only spec for which property (40) holds for arbitrary spec R.

Similarly, we define right-sectioning by:

Definition 41 (Right-Sectioning) For left condition q,

$$\otimes q \quad = \quad I \otimes q$$

□

For which we have the property:

Theorem 42 (Right-Sectioning)

$$\otimes q \circ R \;=\; R \otimes q$$

□

3.2 Symmetry

We define symmetry by:

Definition 43 (Symmetry) $\otimes$ is symmetric iff

$$\forall (R, S :: \; R \otimes S \;=\; S \otimes R)$$

□

There is an other characterization of symmetry:

Theorem 44 (Symmetry) $\otimes$ is symmetric iff

$$\otimes \;=\; \otimes \circ \gamma$$

where γ is the natural transformation such that

$$R \times S \;\circ\; \gamma \;=\; \gamma \;\circ\; S \times R \quad \wedge \quad \gamma > \;=\; I \times I$$

□

3.3 Associativity

We define associativity by:

Definition 45 (Associativity) $\otimes$ is associative iff

$$\forall (R, S, T :: \; R \otimes (S \otimes T) \;=\; (R \otimes S) \otimes T)\,)$$

□

There is another characterization of associativity:

Theorem 46 (Associativity) $\otimes$ is associative iff

$$\otimes \;\circ\; I \times \otimes \;=\; \otimes \;\circ\; \otimes \times I \;\circ\; \beta$$

where β is the natural transformation such that

$$(R \times S) \times T \;\circ\; \beta \;=\; \beta \;\circ\; R \times (S \times T) \quad \wedge \quad \beta > \;=\; I \times (I \times I)$$

□

For the definition of the natural transformations β and γ see [4, 6].

3.4 Idempotency

For idempotency we must be careful. Were we to define idempotency by

$$\forall(R :: \ R \otimes R \ = \ R)$$

we would have in particular $I \otimes I \ = \ I$. Since $I \otimes I \ = \ \otimes \circ I \vartriangle I$ this implies the very strong surjectivity condition $\otimes^{<} \ = \ I$. Fortunately, a simple definition is at hand:

Definition 47 (Idempotency) $\otimes$ is idempotent iff,

$$I \otimes I \quad \sqsubseteq \quad I$$

□

For idempotency, we have the following property

Theorem 48 (Idempotency) $\otimes$ is idempotent iff

$$\forall(f : \ imp.f : \ f \otimes f = (\otimes \circ I \vartriangle I)^{>} \circ f)$$

□

3.5 Units

We define left units by

Definition 49 (Left Unit) $l_{\otimes}$ is a left unit of $\otimes$ iff

$$\begin{array}{ll} & left\_condition.l_{\otimes} \ \wedge \ (\otimes \circ \ll^{\cup})^{>} \ \sqsupseteq \ l_{\otimes}{}^{<} \\ \wedge & \\ & l_{\otimes} \otimes \ = \ (\otimes \circ \gg^{\cup})^{>} \end{array}$$

□

We demand the first conjunct because otherwise $l_{\otimes} \otimes$ is not defined. The second conjunct expresses that $l_{\otimes} \otimes$ should not contain junk outside the domain of the left argument of $\otimes$. The third conjunct expresses that $l_{\otimes}$ is the identity on the domain of the right argument of $\otimes$.

Of course, we can give similar definitions for $r_{\otimes}$ being a right unit of $\otimes$:

Definition 50 (Right Unit) $r_{\otimes}$ is a right unit of $\otimes$ iff,

$$\begin{array}{ll} & left\_condition.r_{\otimes} \ \wedge \ (\otimes \circ \gg^{\cup})^{>} \ \sqsupseteq \ r_{\otimes}{}^{<} \\ \wedge & \\ & \otimes r_{\otimes} \ = \ (\otimes \circ \ll^{\cup})^{>} \end{array}$$

□

For arbitrary binary spec $\otimes$ nothing is known about the uniqueness of the left- or right unit. But if we know that $\otimes$ has a left and a right unit we have:

$$
\begin{array}{ll}
 & r_{\otimes} \\
= & \quad \{ \text{ definition left-, right-unit: } I \sqsupseteq l_{\otimes}\otimes \sqsupseteq r_{\otimes}{<} \ \} \\
 & l_{\otimes}\otimes \circ r_{\otimes} \\
= & \quad \{ \text{ sectioning } \} \\
 & \otimes r_{\otimes} \circ l_{\otimes} \\
= & \quad \{ \text{ definition left-, right-unit: } I \sqsupseteq \otimes r_{\otimes} \sqsupseteq l_{\otimes}{<} \ \} \\
 & l_{\otimes}
\end{array}
$$

So they are the same and hence unique. This leads us to the following definition:

Definition 51 (Unit)
$1_{\otimes}$ is the unique unit of $\otimes$ iff $1_{\otimes}$ is a left and a right unit of $\otimes$.
□

Note: Very often we are interested in the reduce $/(1_{\otimes} \triangledown \otimes)$; that is why we allow ourself to write just $/\otimes$ instead of the cumbersome $/(1_{\otimes} \triangledown \otimes)$.

3.6 Conditionals

Because we now have the notation $\_ \otimes \_$ we can state two more properties about conditionals:

Theorem 52 ($\_ \otimes \_$-Conditional Abides Law)

(a)
$$
\begin{array}{ll}
 & (R \lhd p \rhd S) \otimes (P \lhd p \rhd Q) \\
= & \\
 & (R \otimes P) \lhd p \rhd (S \otimes Q)
\end{array}
$$

(b)
$$
\begin{array}{ll}
 & (R \lhd p \rhd S) \otimes (P \lhd q \rhd Q) \\
= & \\
 & ((R \otimes P) \lhd q \rhd (R \otimes Q)) \lhd p \rhd ((S \otimes P) \lhd q \rhd (S \otimes Q))
\end{array}
$$

□

4 The Boom Hierarchy

The Boom-Hierarchy consists of four levels: trees, lists, bags and sets. A tree can be represented by a bin. Note that many bins can represent the same tree. For instance bin x and $\varepsilon \mathbin{+\!\!+} x$ represent the same tree. Also, we can represent a list by a tree if we forget about the brackets, i.e. $x \mathbin{+\!\!+} (y \mathbin{+\!\!+} z)$ and $(x \mathbin{+\!\!+} y) \mathbin{+\!\!+} z$ represent the same list. A bag can be represented by a list if we forget about the ordering of the elements: $x \mathbin{+\!\!+} y$ and $y \mathbin{+\!\!+} x$ represent the same bag. Similarly, a set can be represented by a bag if we forget about the number of occurrences of an element in a bag: $x \mathbin{+\!\!+} x$ and x represent the same set. In the paper by E. Voermans [24] a construction is given for an equivalence relation on μF such that the equivalence classes of that equivalence relation are just those bins (trees, lists, bags) that represent the same tree (list, bag, set). It is beyond the scope of the current paper to give that construction in detail; only the results are relevant to us. We denote the four equivalence relations by *Tree*, *List*, *Bag* and *Set*. Let $\mathcal{L}$ be one of these relations; we define the following constructors

$$
\begin{array}{lrcl}
(53) & \tau_{\mathcal{L}} & = & \mathcal{L} \circ \tau \\
(54) & \square_{\mathcal{L}} & = & \mathcal{L} \circ \square \\
(55) & +\!\!\!+_{\mathcal{L}} & = & \mathcal{L} \circ +\!\!\!+
\end{array}
$$

So, for instance spec τ_{List} is the singleton-list constructor, i.e. when we apply τ_{List} to an element we get as result the whole equivalence class of *List* of which each bin represents the same singleton-list.

For these three constructors we have the following desired properties:

Theorem 56 (Boom-hierarchy)

$$
\begin{array}{ll}
(a) & +\!\!\!+_{Tree} \circ \varepsilon_{Tree} \vartriangle Tree \;=\; Tree \\
(b) & +\!\!\!+_{Tree} \circ Tree \vartriangle \varepsilon_{Tree} \;=\; Tree \\
(c) & +\!\!\!+_{List} \circ +\!\!\!+_{List} \times I \;=\; +\!\!\!+_{List} \circ I \times +\!\!\!+_{List} \circ \beta \\
(d) & +\!\!\!+_{Bag} \;=\; +\!\!\!+_{Bag} \circ \gamma \\
(e) & +\!\!\!+_{Set} \circ I \vartriangle I \;=\; Set
\end{array}
$$

where β and γ are the natural transformations as used for the definition of associativity (46) and symmetry (44).
$\square$

For $\mathcal{L}$ equal to *Tree*, *List*, *Bag* or *Set* we have the so-called absorption rule:

$$(57) \quad \mathcal{L} \circ +\!\!\!+ \;=\; \mathcal{L} \circ +\!\!\!+ \circ \mathcal{L} \times \mathcal{L} \;=\; \mathcal{L} \circ +\!\!\!+ \circ I \times \mathcal{L} \;=\; \mathcal{L} \circ +\!\!\!+ \circ \mathcal{L} \times I$$

By unfolding the definition of $+\!\!\!+_{\mathcal{L}}$ and $\varepsilon_{\mathcal{L}}$ and using the absorption rule, we can shift the type information entirely to the left, which gives us:

Theorem 58 (Property constructors)

$$
\begin{array}{ll}
(a) & Tree \circ \varepsilon +\!\!\!+ \;=\; Tree \\
(b) & Tree \circ +\!\!\!+ \varepsilon \;=\; Tree \\
(c) & List \circ +\!\!\!+ \circ +\!\!\!+ \times I \;=\; List \circ +\!\!\!+ \circ I \times +\!\!\!+ \circ \beta \\
(d) & Bag \circ +\!\!\!+ \circ \gamma \;=\; Bag \circ +\!\!\!+ \\
(e) & Set \circ +\!\!\!+ \circ I \vartriangle I \;=\; Set
\end{array}
$$

where β and γ are the natural transformations as used for the definition of associativity (46) and symmetry (44).
$\square$

For example, property (d) expresses the fact that bins constructed by the functions $+\!\!\!+$ and $+\!\!\!+ \circ \gamma$ represent the same bag. In general it is the case that if

$$\mathcal{L} \circ R \;=\; \mathcal{L} \circ S$$

then R and S are the same up to the choice of the representatives of the equivalence classes resulting from the application of R and S to an element.

The properties of the constructors take on a more familiar form if we compose both left and right sides with an appropriate number of argument specs. For example, property (a) becomes, for all specs R,

$$(59)\quad \mathit{Tree} \circ \varepsilon +\!\!\!+ R \;=\; \mathit{Tree} \circ R$$

and similarly property (b) becomes

$$(60)\quad \mathit{Tree} \circ R +\!\!\!+ \varepsilon \;=\; \mathit{Tree} \circ R$$

If we compose both left and right sides of (e) with imp f, property (e) becomes, for all imps f,

$$(61)\quad \mathit{Set} \circ f +\!\!\!+ f \;=\; \mathit{Set} \circ f$$

If more bins represent the same element of a particular type, the equivalence classes become bigger. Formally stated, we have the following lattice ordering which follows from the fact that the four types are the least solutions of a specification and that each type higher in the hierarchy satisfies the conditions of the types lower in the hierarchy:

$$(62)\quad \mu F \sqsubseteq \mathit{Tree} \sqsubseteq \mathit{List} \sqsubseteq \mathit{Bag} \sqsubseteq \mathit{Set}$$

Also, we have the following properties:

$$\begin{array}{ll}
(63) & \mathit{Tree} = \mathit{Tree} \circ \mu F \\
(64) & \mathit{List} = \mathit{List} \circ \mathit{Tree} = \mathit{List} \circ \mu F \\
(65) & \mathit{Bag} = \mathit{Bag} \circ \mathit{List} = \mathit{Bag} \circ \mathit{Tree} = \mathit{Bag} \circ \mu F \\
(66) & \mathit{Set} = \mathit{Set} \circ \mathit{Bag} = \mathit{Set} \circ \mathit{List} = \mathit{Set} \circ \mathit{Tree} = \mathit{Set} \circ \mu F
\end{array}$$

We say that X *respects* $\mathcal{L}$ if

$$X \circ \mathcal{L} \;=\; X$$

this condition expresses that spec X does not differentiate between representatives of the same equivalence class of $\mathcal{L}$. For instance, if $R \circ \mathit{Set} = R$ holds then spec R yields the same result when applied to bins x and $x +\!\!\!+ x$.

For each level of the Boom-hierarchy we have the following conditions for a catamorphism to be of that type.

Theorem 67 (Type of Catamorphism)
In order to state the theorem let us introduce the following abbrevations. First let $\mathcal{C}$ be a shorthand for $(\!|R \triangledown (S \triangledown \otimes)|\!)$. Then let us consider the following five properties, which we refer to later by their labels:

$$\begin{array}{ll}
(a) & \otimes \circ (S \circ \top\!\!\top) \vartriangle I \circ \mathcal{C} = \mathcal{C} \\
(b) & \otimes \circ I \vartriangle (S \circ \top\!\!\top) \circ \mathcal{C} = \mathcal{C} \\
(c) & (\mathcal{C} \underline{\otimes} \mathcal{C}) \underline{\otimes} \mathcal{C} = \mathcal{C} \underline{\otimes} (\mathcal{C} \underline{\otimes} \mathcal{C}) \circ \beta \\
(d) & \mathcal{C} \underline{\otimes} \mathcal{C} = \mathcal{C} \underline{\otimes} \mathcal{C} \circ \gamma \\
(e) & \otimes \circ \mathcal{C} \vartriangle \mathcal{C} = \mathcal{C}
\end{array}$$

(Note that (c), respectively (d), holds if $\otimes$ is associative, respectively symmetric).

Now we can state the theorem, which is:

$\mathcal{C}$ respects *Tree* $\equiv$ (a) $\wedge$ (b).
$\mathcal{C}$ respects *List* $\equiv$ $\mathcal{C}$ respects *Tree* $\wedge$ (c).
$\mathcal{C}$ respects *Bag* $\equiv$ $\mathcal{C}$ repects *List* $\wedge$ (d).
$\mathcal{C}$ respects *Set* $\equiv$ $\mathcal{C}$ respects *Bag* $\wedge$ (e).
$\square$

4.1 Map and Filter

In this section we derive properties for map and filter concerning laws. From these properties it follows that the type of a catamorphism remains the same if it is composed with a filter or a map (in the case of *Set* only if the argument of the map is an imp). An important theorem is:

Theorem 68 (Type-η Fusion) For $\mathcal{L}$ equal to *Tree*, *List*, *Bag* or *Set*,

$$\mathcal{L} \circ (\!|R \triangledown \eta|\!) \;=\; (\!|(\mathcal{L} \circ R) \triangledown (\mathcal{L} \circ \eta)|\!)$$

$\square$

Using this property we can prove the following type judgements:

Theorem 69 For $\mathcal{L}$ equal to *Tree*, *List* or *Bag*, and imp f,

(a) $\mathcal{L} \circ (\!|R \triangledown \eta|\!)$ respects $\mathcal{L}$
(b) $\mathit{Set} \circ (\!|f \triangledown \eta|\!)$ respects Set
(c) $\mathcal{L} \circ *R$ respects $\mathcal{L}$
(d) $\mathit{Set} \circ *f$ respects Set
(e) $\mathcal{L} \circ \triangleleft p$ respects $\mathcal{L}$
(f) $\mathit{Set} \circ \triangleleft p$ respects Set
$\square$

The importance of theorem (69) is that if a spec R does not differentiate between representatives of, say, the same bag, i.e. spec R respects *Bag*, then for instance $R \circ \triangleleft p$ also respects *Bag*. Thus using these theorems we can move around the type information in our formulae.

5 Comparison with Quantifier Notation

In this section we will make a link to the Quantifier Notation. We define

(70) $\otimes(i : p.i : R.i) \;=\; /\!\otimes \circ *R \circ \triangleleft p$

where $R.i$ is defined by

(71) $R.i \;=\; R \circ i\bullet$

with $i\bullet$ being a point "pointing at i", i.e. $i\bullet< \; = \; \{\,(i,i)\,\}$.

It is vital to note that the expression $\otimes(i\ :\ p.i\ :\ R.i)$ denotes a function. This appears to be counter to normal usage; that it is not so is explained by the fact that the domain of the dummy i is always left implicit in the quantifier notation. We can make it explicit by writing $\otimes(i\ \in\ A\ :\ p.i\ :\ R.i)$; if we now suppose that $\mathcal{A}$ is an enumeration of the elements of the type A and $\otimes$ is both associative and symmetric then we make the definition

(72) $\otimes(i\ \in\ A : p.i : R.i) \;=\; /\otimes \circ *R \circ \triangleleft p \circ \mathcal{A}$

Note that for the quantification over $\otimes$ to be meaningful it is always assumed that $\otimes$ is both associative and symmetric. We will not demand this unless it is necessary.

Two rules known for the quantifier notation hold at the level of bins. The first rule is trading; the laws govern the interchange of expressions between the range and the function part of the quantification. Using theorem (36a) and (30) it is easy to prove that:

(73) $/\otimes \circ *R \circ \triangleleft p \;=\; /\otimes \circ *(R \lhd p \rhd 1_{\otimes})$

The well-known trading rules for universal and existential quantification are immediate corollaries of (73).

(74) $/\mathbf{and} \circ *\overline{q} \circ \triangleleft p \;=\; /\mathbf{and} \circ *(\overline{p \Rightarrow q})$

(75) $/\mathbf{or} \circ *\overline{q} \circ \triangleleft p \;=\; /\mathbf{or} \circ *(\overline{p \wedge q})$

where p and q are right-conditions, and $\overline{q}$ is a total imp to $\mathbb{B}$ defined by

$$\overline{q} \;=\; \mathbf{true} \lhd q \rhd \mathbf{false}$$

with **true** and **false** the points "pointing" at true and false ($\mathbb{B}$ is a monotype containing two elements: false and true) and **and** and **or** the total imps from $\mathbb{B} \times \mathbb{B}$ to $\mathbb{B}$ corresponding to disjunction and conjunction.

To derive (74) and (75) it suffices to observe that

(76) $\overline{q} \lhd p \rhd \mathbf{true} \;=\; \overline{p \Rightarrow q}$

and

(77) $\overline{q} \lhd p \rhd \mathbf{false} \;=\; \overline{p \wedge q}$

which follows directly from the definition of the bar and the properties of conditionals. As an example we prove (76):

$$\begin{array}{ll} & \overline{q} \lhd p \rhd \mathbf{true} \\ = & \quad \{\ \text{property conditionals (33d)}\ \} \\ & \mathbf{true} \lhd \neg p \rhd \overline{q} \\ = & \quad \{\ \text{definition bar}\ \} \\ & \mathbf{true} \lhd \neg p \rhd (\mathbf{true} \lhd q \rhd \mathbf{false}) \end{array}$$

$$
\begin{array}{ll}
= & \quad \{ \text{ property conditionals (33g) } \} \\
& \mathbf{true} \lhd \neg p \vee q \rhd \mathbf{false} \\
= & \quad \{ \text{ definition } \Rightarrow\text{, definition bar } \} \\
& \overline{p \Rightarrow q}
\end{array}
$$

Less familiar consequences of (73) are the trading rules for equivalence and inequivalence.

(78) $/\!\equiv \circ *\overline{q} \circ \lhd p \;=\; /\!\equiv \circ *(\overline{p \Rightarrow q})$
(79) $/\!\not\equiv \circ *\overline{q} \circ \lhd p \;=\; /\!\not\equiv \circ *(\overline{p \wedge q})$

These follow because, like conjunction, the unit of equivalence is **true** and, like disjunction, the unit of inequivalence is **false**.

The literal translation of (74) and (75) into the quantifier notation yields

$$\forall(i : \; p.i : \; q.i) \;=\; \forall(i :: \; p.i \Rightarrow q.i)$$

and

$$\exists(i : p.i : q.i) \;=\; \exists(i :: p.i \wedge q.i)$$

The trading rules in the quantifier calculus are, however, slightly more general; specifically:

$$\forall(i : \; p.i \wedge r.i : \; q.i) \;=\; \forall(i : \; r.i : \; p.i \Rightarrow q.i)$$

and

$$\exists(i : p.i \wedge r.i : q.i) \;=\; \exists(i : r.i : p.i \wedge q.i)$$

But these properties stated in our formalism follow directly from theorem (37):

(80) $\lhd(p \wedge r) \;=\; \lhd p \circ \lhd r$

The second rule we have already derived is range translation. From the filter range translation rule (38) and compositionality of map it follows that, for imp f,

(81) $/\!\otimes \circ *R \circ \lhd p \circ *f \;=\; /\!\otimes \circ *(R \circ f) \circ \lhd(p \circ f) \circ *f_{>}$

In the quantifier notation this would be expressed as

(82) $\otimes(i \in t.A : p.i : f.i) \;=\; \otimes(j \in A : p.(t.j) : f.(t.j))$

In the following sections we derive the other well-known rules like the unit rule, range splitting and range disjunction. We try to derive these properties as early as possible in the Boom-hierarchy.

5.1 Unit Law

In this section we concentrate on trees. Using the results of the previous section and the property of the constructors (58a) we can prove the unit rule. We have the following property:

Theorem 83 (Unit)

$$Tree \circ \lhd \mathit{false} = Tree \circ \varepsilon \circ \mu F$$

□

Note that the added type information is crucial: without it, the equality does not hold. For instance, $\lhd \mathit{false}$ transforms $a +\!\!\!+ (b +\!\!\!+ c)$ into $\varepsilon +\!\!\!+ (\varepsilon +\!\!\!+ \varepsilon)$ and the latter is only equal to ε if we consider trees. The added "$\circ\ \mu F$" is necessary because the right domain of $\lhd \mathit{false}$ is μF.

Using theorem (83) we can prove:

Corollary 84 (Unit Rule) If $(\!| R \triangledown (S \triangledown \otimes) |\!)$ respects $Tree$ then

$$(\!| R \triangledown (S \triangledown \otimes) |\!) \circ \lhd \mathit{false} = S \circ \top\!\!\top \circ \mu F$$

Proof

$$\begin{array}{cl}
 & (\!| R \triangledown (S \triangledown \otimes) |\!) \circ \lhd \mathit{false} \\
= & \qquad \{\ (\!| R \triangledown (S \triangledown \otimes) |\!) \text{ respects } Tree\ \} \\
 & (\!| R \triangledown (S \triangledown \otimes) |\!) \circ Tree \circ \lhd \mathit{false} \\
= & \qquad \{\ \text{theorem (83)}\ \} \\
 & (\!| R \triangledown (S \triangledown \otimes) |\!) \circ Tree \circ \varepsilon \circ \mu F \\
= & \qquad \{\ (\!| R \triangledown (S \triangledown \otimes) |\!) \text{ respects } Tree\ \} \\
 & (\!| R \triangledown (S \triangledown \otimes) |\!) \circ \varepsilon \circ \mu F \\
= & \qquad \{\ \text{definition } \varepsilon\ \} \\
 & (\!| R \triangledown (S \triangledown \otimes) |\!) \circ \Box \circ \top\!\!\top \circ \mu F \\
= & \qquad \{\ \text{computation rule } \Box\ \} \\
 & S \circ \top\!\!\top \circ \mu F
\end{array}$$

□

The above calculation is a nice example of how we treat type information in our system. By the assumption the type information pops up at the right place and having used the type information we can get rid of it by using the assumption again.

Examples of the unit rule are

$$\begin{array}{lcl}
/+ \circ *R \circ \lhd \mathit{false} & = & 0\bullet \circ \mu F \\
/\times \circ *R \circ \lhd \mathit{false} & = & 1\bullet \circ \mu F \\
/\mathbf{and} \circ *R \circ \lhd \mathit{false} & = & \mathbf{true} \circ \mu F \\
/\mathbf{or} \circ *R \circ \lhd \mathit{false} & = & \mathbf{false} \circ \mu F.
\end{array}$$

because $0\bullet$, $1\bullet$, **true** and **false** are the units of $+$, $\times$, **and** and **or**. These are written in the quantifier form as follows

$$\begin{array}{rcl}\Sigma(i : \mathit{false} : R.i) & = & 0 \\ \Pi(i : \mathit{false} : R.i) & = & 1 \\ \forall(i : \mathit{false} : R.i) & = & \mathit{true} \\ \exists(i : \mathit{false} : R.i) & = & \mathit{false}\end{array}$$

Note that the proof given here is valid for trees (and thus also if the catamorphism respects lists, bags or sets). This is an improvement over the one in [2] since there the range splitting rule was used but that rule is only valid for bags and hence also for sets.

5.2 Associativity and Symmetry

Throughout this section we assume the existence of an associative and symmetric binary spec $\otimes$ and the existence of a spec U, such that

$$U \;=\; U \otimes U$$

An obvious candidate for U is $1_\otimes$ if it exists.

First we prove a property of $\otimes$:

Theorem 85 ($\underline{\otimes}$-$\otimes$ Abides Law)

$$(R \underline{\otimes} S) \otimes (P \underline{\otimes} Q) \;=\; (R \otimes P) \underline{\otimes} (S \otimes Q)$$

Proof

$$\begin{array}{cl} & (R \underline{\otimes} S) \;\otimes\; (P \underline{\otimes} Q) \\ = & \quad \{\ \text{property (22)}\ \} \\ & ((R \circ \ll) \otimes (S \circ \gg)) \otimes ((P \circ \ll) \otimes (Q \circ \gg)) \\ = & \quad \{\ \otimes \text{ associative and symmetric }\ \} \\ & ((R \circ \ll) \otimes (P \circ \ll)) \otimes ((S \circ \gg) \otimes (Q \circ \gg)) \\ = & \quad \{\ \ll, \gg \text{ imps, split-imp fusion (23)}\ \} \\ & (R \otimes P \circ \ll) \;\otimes\; (S \otimes Q \circ \gg) \\ = & \quad \{\ \text{property (22)}\ \} \\ & (R \otimes P) \underline{\otimes} (S \otimes Q) \end{array}$$

□

For this spec $\otimes$ and U we can prove:

Theorem 86 (Asso-Sym Rule)

$$(\!|(R \otimes S) \triangledown (U \triangledown \otimes)|\!) \;=\; (\!|R \triangledown (U \triangledown \otimes)|\!) \;\otimes\; (\!|S \triangledown (U \triangledown \otimes)|\!)$$

□

We can now prove, because a filter is an imp:

Theorem 87

$$\begin{array}{cl} & (\!|(R \otimes S) \triangledown (U \triangledown \otimes)|\!) \;\circ\; \triangleleft p \\ = & \\ & ((\!|R \triangledown (U \triangledown \otimes)|\!) \;\circ\; \triangleleft p) \;\otimes\; ((\!|S \triangledown (U \triangledown \otimes)|\!) \;\circ\; \triangleleft p) \end{array}$$

Proof

$$\begin{array}{cl}
 & (\!|(R \otimes S) \triangledown (U \triangledown \otimes)|\!) \circ \triangleleft p \\
= & \quad \{ \text{ theorem (86) } \} \\
 & (\!|R \triangledown (U \triangledown \otimes)|\!) \otimes (\!|S \triangledown (U \triangledown \otimes)|\!) \circ \triangleleft p \\
= & \quad \{ \triangleleft p \text{ imp, split-imp fusion (23) } \} \\
 & ((\!|R \triangledown (U \triangledown \otimes)|\!) \circ \triangleleft p) \otimes ((\!|S \triangledown (U \triangledown \otimes)|\!) \circ \triangleleft p)
\end{array}$$

□

Theorem (87) is the direct analogue of the associativity and symmetry rule in [2], expressed in the quantifier notation as follows:

$$\otimes(i : p.i : f.i \otimes g.i) = \otimes(i : p.i : f.i) \otimes \otimes(i : p.i : g.i)$$

By definition $Bag \circ +\!\!\!+$ is associative and $Bag \circ \varepsilon$ is the unit of $Bag \circ +\!\!\!+$, thus theorem (86) holds with $U := Bag \circ \square$ and $\otimes := Bag \circ +\!\!\!+$. This results in:

Theorem 88 (Asso-Sym Rule)

$$Bag \circ (\!|(R +\!\!\!+ S) \triangledown \eta|\!) = Bag \circ (\!|R \triangledown \eta|\!) +\!\!\!+ (\!|S \triangledown \eta|\!)$$

□

For the proof of Range Splitting and Range Disjunction in the next section we will use theorem (88) to move the join operator inside the catamorphism, then we use property (52a) or (52b) to move it inside the arguments of the conditionals and finally, with some shuffling around, we use the properties of the constructors to remove it entirely.

Theorem 89 (Range Splitting)

$$Bag \circ \triangleleft p +\!\!\!+ \triangleleft \neg p = Bag$$

Proof

First we prove:

$$\begin{array}{cl}
 & Bag \circ (\tau \lhd p \rhd \varepsilon) +\!\!\!+ (\tau \lhd \neg p \rhd \varepsilon) \\
= & \quad \{ \text{ property conditionals (33d) } \} \\
 & Bag \circ (\tau \lhd p \rhd \varepsilon) +\!\!\!+ (\varepsilon \lhd p \rhd \tau) \\
= & \quad \{ +\!\!\!+ \text{ abides with <p> (52a) } \} \\
 & Bag \circ (\tau +\!\!\!+ \varepsilon) \lhd p \rhd (\varepsilon +\!\!\!+ \tau) \\
= & \quad \{ \text{ property conditonals (33j) } \} \\
 & (Bag \circ \tau +\!\!\!+ \varepsilon) \lhd p \rhd (Bag \circ \varepsilon +\!\!\!+ \tau) \\
= & \quad \{ \text{ property constructors (58ab) } \} \\
 & (Bag \circ \tau) \lhd p \rhd (Bag \circ \tau) \\
= & \quad \{ \text{ property conditionals (33a) } \} \\
 & Bag \circ \tau
\end{array}$$

And from this the theorem follows:

$$
\begin{array}{ll}
 & Bag \circ \triangleleft p ⧺ \triangleleft \neg p \\
= & \quad \{ \text{ definition filter } \} \\
 & Bag \circ (\!|(\tau \lhd p \rhd \varepsilon) \triangledown \eta|\!) ⧺ (\!|(\tau \lhd \neg p \rhd \varepsilon) \triangledown \eta|\!) \\
= & \quad \{ \text{ asso-sym rule (88) } \} \\
 & Bag \circ (\!|((\tau \lhd p \rhd \varepsilon) ⧺ (\tau \lhd \neg p \rhd \varepsilon)) \triangledown \eta|\!) \\
= & \quad \{ \text{ type-}\eta \text{ fusion (68) } \} \\
 & (\!|(Bag \circ (\tau \lhd p \rhd \varepsilon) ⧺ (\tau \lhd \neg p \rhd \varepsilon)) \triangledown (Bag \circ \eta)|\!) \\
= & \quad \{ \text{ calculation above } \} \\
 & (\!|(Bag \circ \tau) \triangledown (Bag \circ \eta)|\!) \\
= & \quad \{ \text{ type-}\eta \text{ fusion (68) } \} \\
 & Bag \circ (\!|\tau \triangledown \eta|\!) \\
= & \quad \{ (\!|\tau \triangledown \eta|\!) = \mu F, Bag \circ \mu F = Bag \} \\
 & Bag
\end{array}
$$

□

Using the range splitting rule, we can prove:

Theorem 90 If $(\!|R \triangledown (S \triangledown \otimes)|\!)$ respects Bag then,

$$
\begin{array}{ll}
 & (\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft q \\
= & \\
 & ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft (p \wedge q)) \otimes ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft (\neg p \wedge q))
\end{array}
$$

Proof

$$
\begin{array}{ll}
 & (\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft q \\
= & \quad \{ (\!|R \triangledown (S \triangledown \otimes)|\!) \text{ respects } Bag\text{, range splitting } \} \\
 & (\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft p ⧺ \triangleleft \neg p \circ \triangleleft q \\
= & \quad \{ \text{ definition } \_⧺\_\text{, computation rule } ⧺ \} \\
 & ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft p) \otimes ((\!|R \triangledown S \triangledown \otimes|\!) \circ \triangleleft \neg p) \circ \triangleleft q \\
= & \quad \{ \triangleleft q \text{ imp, split-imp fusion } \} \\
 & ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft p \circ \triangleleft q) \otimes ((\!|R \triangledown S \triangledown \otimes|\!) \circ \triangleleft \neg p \circ \triangleleft q) \\
= & \quad \{ \text{ filter-distribution } \} \\
 & ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft (p \wedge q)) \otimes ((\!|R \triangledown S \triangledown \otimes|\!) \circ \triangleleft (\neg p \wedge q))
\end{array}
$$

□

Expressed in the quantifier notation, theorem (90) takes the form:

$$
\begin{array}{ll}
 & \otimes(i : q.i : f.i) \\
= & \\
 & \otimes(i : p.i \wedge q.i : f.i) \otimes \otimes(i : \neg p.i \wedge q.i : f.i)
\end{array}
$$

5.3 Adding Idempotency

In this section we add idempotency, i.e. we consider sets. We know that $Set = Set \circ Bag$ so we may use the asso-sym rule (88). Furthermore, we know that $Set \circ ⧺$ is idempotent, which gives us:

Theorem 91 (Range Disjunction)

$$Set \circ \triangleleft p \mathbin{+\!\!\!+} \triangleleft q \;=\; Set \circ \triangleleft (p \vee q)$$

Proof

First we prove:

$$\begin{array}{ll}
 & Set \circ (\tau \lhd p \rhd \varepsilon) \mathbin{+\!\!\!+} (\tau \lhd q \rhd \varepsilon) \\
= & \quad \{ \text{ property conditionals (52b) } \} \\
 & Set \circ ((\tau \mathbin{+\!\!\!+} \tau) \lhd q \rhd (\tau \mathbin{+\!\!\!+} \varepsilon)) \lhd p \rhd ((\varepsilon \mathbin{+\!\!\!+} \tau) \lhd q \rhd (\varepsilon \mathbin{+\!\!\!+} \varepsilon)) \\
= & \quad \{ \text{ property conditionals (33j), constructors (58abe) } \} \\
 & Set \circ (\tau \lhd q \rhd \tau) \lhd p \rhd (\tau \lhd q \rhd \varepsilon) \\
= & \quad \{ \text{ property conditionals (33a) } \} \\
 & Set \circ \tau \lhd p \rhd (\tau \lhd q \rhd \varepsilon) \\
= & \quad \{ \text{ property conditionals (33g) } \} \\
 & Set \circ \tau \lhd p \vee q \rhd \varepsilon
\end{array}$$

Then we have:

$$\begin{array}{ll}
 & Set \circ \triangleleft p \mathbin{+\!\!\!+} \triangleleft q \\
= & \quad \{ \text{ definition filter } \} \\
 & Set \circ (\!|\tau \lhd p \rhd \varepsilon \triangledown \eta|\!) \mathbin{+\!\!\!+} (\!|\tau \lhd q \rhd \varepsilon \triangledown \eta|\!) \\
= & \quad \{ \text{ theorem (88) } \} \\
 & Set \circ (\!|((\tau \lhd p \rhd \varepsilon) \mathbin{+\!\!\!+} (\tau \lhd q \rhd \varepsilon)) \triangledown \eta|\!) \\
= & \quad \{ \text{ theorem (68), calculation above } \} \\
 & Set \circ (\!|\tau \lhd p \vee q \rhd \varepsilon \triangledown \eta|\!) \\
= & \quad \{ \text{ definition filter } \} \\
 & Set \circ \triangleleft (p \vee q)
\end{array}$$

□

And again we can combine this with a catamorphism:

Theorem 92 If $(\!|R \triangledown (S \triangledown \otimes)|\!)$ respects Set then,

$$\begin{array}{ll}
 & (\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft (p \vee q) \\
= & \\
 & ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft p) \otimes ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft q)
\end{array}$$

Proof

$$\begin{array}{ll}
 & (\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft (p \vee q) \\
= & \quad \{ \; (\!|R \triangledown (S \triangledown \otimes)|\!) \text{ respects } Set\text{, range disjunction } \} \\
 & (\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft p \mathbin{+\!\!\!+} \triangleleft q \\
= & \quad \{ \text{ definition } _\mathbin{+\!\!\!+}_\text{, computation rule } \mathbin{+\!\!\!+} \} \\
 & ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft p) \otimes ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \triangleleft q)
\end{array}$$

□

Theorem (92) stated in the quantifier notation yields:

$$\otimes(i : p.i \vee q.i : f.i) \;=\; \otimes(i : p.i : f.i) \;\otimes\; \otimes(i : q.i : f.i)$$

6 Conclusion

By now the so-called "Boom hierarchy of types" is very familiar (if perhaps not under that name) and the substantial majority of properties established in this paper have been published elsewhere (in texts on APL, in Backus's Turing award lecture and in Meertens' paper). The contribution of this paper has principally been to organise the rules, making clear at which level of the hierarchy each rule becomes valid. The paper has also generalised the rules from a strictly typed, functional framework to a polymorphic, relational framework. Two surprising outcomes are the absence of appeals to extensionality and the low incidence of properties that rely on functionality.

A major distinguishing feature of the spec calculus is that it tries to capture *true* polymorphism rather than *parameterised* polymorphism, which is expressed by naturality properties in category theory. A consequence of this design decision is that type considerations occasionally enter into equational laws rather than being expressed in some overall context within which the law is applicable. This difference is illustrated by the split operator. In category theory the arrow $R \vartriangle S$ is only defined if R and S have the same right domain; if this is the case one has the computation rule

(93) $\ll \;\circ\; R \vartriangle S \;=\; R$

In the spec calculus split is a *total* operator: $R \vartriangle S$ is a spec for all specs R and S irrespective of whether R and S have the same right domain, and the rule (93) takes the form

(94) $\ll \;\circ\; R \vartriangle S \;=\; R \;\circ\; S{>}$

Our experience is that this has been a fortuitious design decision: one is rarely hindered by the types that occasionally crop up in calculations, and calculations proceed more smoothly because one is not continually nervous about unconscious omission of some type restriction, the laws themselves containing the reminder that such is necessary.

This aspect of the theory is particularly emphasised in this paper: a great many of our calculations are prefaced by "$\mathit{List} \circ$" or "$\mathit{Bag} \circ$" etc. It may indeed seem that the decision to develop a calculus in which types are part and parcel of the equational laws was misguided: here, after all, is the evidence how cumbersome it can be. But one must remember the whole content of this paper is the discussion of which laws are valid for Trees, which for Lists etc. and it is because of this that the type constraints are ever present.

7 Acknowledgements

We would like to thank Roland Backhouse, Ed Voermans and Netty van Gasteren for their contributions to this work.

Preparation of this paper was expedited by the use of the proof editor developed by Paul Chisholm [14].

References

1. R.C. Backhouse. An exploration of the Bird-Meertens formalism. Technical Report CS 8810, Department of Computing Science, Groningen University, 1988.
2. R.C. Backhouse. Program construction and verification. Prentice-Hall, 1986.
3. Roland Backhouse and Paul Hoogendijk. Elements of a Relational Theory of Datatypes. To appear in: *Proceedings of IFIP summerschool*, Brazil 1992.
4. R.C. Backhouse, P. de Bruin, P. Hoogendijk, G. Malcolm, T.S. Voermans, and J. van der Woude. Polynomial relators. To appear in: *Proceedings of the 2nd Conference on Algebraic Methodology and Software Technology*, May 22-25, 1991.
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. R.C. Backhouse, T.S. Voermans, J. v.d. Woude. A Relational Theory Of Datatypes. *Proceedings of the EURICS on Calculational Theories of Program Structure, Ameland, The Netherlands*, September 1991.
7. 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, 1978.
8. R.S. Bird. Transformational Programming and the Paragraph Problem. *Science of Computing Programming,* 6:159-189, 1986.
9. 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.
10. R.S. Bird. Lectures on constructive functional programming. Oxford University, 1988.
11. R.S. Bird and 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.
12. 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.
13. 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.
14. P. Chisholm. Calculation by Computer. In *Third International Workshop Software Engineering and its Applications*, pages 713-728, Toulouse, France, December 3-7 1990. EC2.
15. E.W. Dijkstra, W.H.J. Feijen. Een methode van programmeren. Academic Service, 1984.
16. C.A.R. Hoare. A couple of novelties in the propositional calculus. *Zeitschr. fuer Math. Logik und Grundlagen der Math.*, 31(2):173-178, 1985.
17. C.A.R. Hoare *et al.* Laws of Programming. *Communications of the ACM*, 30(8):672-686, 1987.
18. P.F. Hoogendijk. A Hierarchy of Freebies. *Proceedings of the EURICS on Calculational Theories of Program Structure, Ameland, The Netherlands*, September 1991.
19. P.F. Hoogendijk. The Boom hierarchy. *Proceedings of the EURICS on Calculational Theories of Program Structure, Ameland, The Netherlands*, September 1991.

20. K. Iverson. A Programming Language. John Wiley & Sons New York, 1962.
21. G. Malcolm. Algebraic data types and program transformation. PhD thesis, Groningen University, 1990.
22. 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.
23. G. Schmidt and T. Ströhlein. Relationen und Grafen. Springer-Verlag, 1988.
24. E. Voermans, Pers as Types, Inductive Types and Types with Laws. To appear in: *PHOENIX seminar and workshop on declarative programming, Sasbachwalden 1991.* Workshops in computing, Springer-Verlag.

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