ELSEVIER Science of Computer Programming 22 (1994) 67–105

Science of Computer Programming

Relational programming laws in the tree, list, bag, set hierarchy

Paul F. Hoogendijk*, Roland C. Backhouse

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

Communicated by C. Morgan; revised October 1993

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" for the construction of functional programs and the "Dijkstra–Feijen calculus" for the construction of imperative programs.

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 [23] and has been further refined and applied in a number of papers by Bird and Meertens [10–12,15,16]. Its beauty derives from the small scale of the theory itself compared with the large scale of its 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 [8]; Iverson [21] 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"[1] after H.J. Boom to whom Meertens attributes the concept.

* Corresponding author. E-mail: {paulh, rolandb}@win.tue.nl.

[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.

SSDI 0167-6423(93)E0014-O

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 [7] 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 characterized 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 [7].

The unconventional nature (and perhaps also the conciseness) of the notations used in the Bird–Meertens formalism makes the formalism difficult to comprehend for some 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 in 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. 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 [3] 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 [1,4]. 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 Boom hierarchy in our system and we show 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.

There is some, unavoidable, overlap between this paper and [7], the overlap occurring within Sections 2 and 3. The novice reader would be well-advised to begin by reading [7] before embarking on this paper, in particular since Sections 2 and 3 of this paper form a very concise summary of [7].

2. The algebraic framework

The relational theory of datatypes we exploit combines a (point-free) axiomatisation of the algebra of relations with axioms postulating the existence of a unit type, a sum and a product operator. The axiom system and basic rules are summarised in this section. Full details can be found in [1,4,5].

2.1. Relation algebra

We begin with the axiomatisation of the algebra of relations. 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, [26].

2.1.1. Plat calculus

Let $\mathscr{A}$ 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 $\mathscr{A}$ we impose the structure of a complete, completely distributive, complemented lattice $(\mathscr{A}, \sqcap, \sqcup, \neg, \top\top, \bot\bot)$ 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. 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: $(\mathscr{A}, \circ, I)$ 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. That is, for $V, W \subseteq \mathscr{A}$, $(\sqcup V) \circ (\sqcup W) = \sqcup(R, S: R \in V \wedge S \in W: R \circ S)$.

The third layer is the reverse structure: $(\mathscr{A}, \cup)$ where $\cup$ is a unary post-fix operator such that it is its own inverse. The interface with the first layer is that $\cup$ is an isomorphism of plats. That is, for all $R, S \in \mathscr{A}$, $R \cup \sqsupseteq S \equiv R \sqsupseteq S \cup$. 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. The constants $\top\top$, $\bot\bot$, and I are, respectively, the universal relation, the empty relation and the identity relation. The operator $\circ$ is relational composition and $\cup$ is the converse operator.

2.1.2. 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.

2.1.3. The Middle Exchange Rule and the Cone Rule

To the above axioms we now add an axiom relating all three layers:

Middle Exchange Rule. $X \sqsupseteq R \circ Y \circ S \equiv \neg Y \sqsupseteq R \cup \circ \neg X \circ S \cup$.

In a system with negation, this axiom is equivalent to the rule variously known as the Dedekind rule [25] or the law of modularity [18]: $(R \sqcap T \circ S \cup) \circ S \sqsupseteq R \circ S \sqcap T$.

Our last axiom, which is sometimes referred to as Tarski's Rule, we call the

Cone Rule. $\top\top \circ R \circ \top\top = \top\top \equiv R \neq \bot\bot$.

2.2. Imps and domains

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 2.1.

(a) R is an *imp* iff $I \sqsupseteq R \circ R\cup$.

(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.

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 as follows.

Definition 2.2 (*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 equation:

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

One of the main considerations of the development of the theory described in [1] 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, type considerations being a part of a particular law itself rather than being expressed in some overall context within which the law is applicable.

2.3. 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 [1,4] an extension to the notion of functor is given, the so-called "relator":

Definition 2.3. 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, i.e. $F.(R>) = (F.R)>$. Another is that relators respect imps.

In this section we postulate axioms guaranteeing the existence of a sum relator, a product relator, and a unit monotype. The so-called map relators (of which trees and lists are an instance) are then defined via fixed-points with these as building blocks.

2.3.1. Sum and product

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

$$P \vartriangle Q = (\ll \cup \circ P) \sqcap (\gg \cup \circ Q), \tag{1}$$

$$P \triangledown Q = (P \circ \subset\!\!\rightarrow \cup) \sqcup (Q \circ \leftarrow\!\!\supset \cup), \tag{2}$$

$$P \times Q = (P \circ \ll) \vartriangle (Q \circ \gg), \tag{3}$$

$$P + Q = (\subset\!\!\rightarrow \circ P) \triangledown (\leftarrow\!\!\supset \circ Q). \tag{4}$$

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 $\subset\!\!\rightarrow$ and $\leftarrow\!\!\supset$ mapping the term x to the terms $\subset\!\!\rightarrow.x$ and $\leftarrow\!\!\supset.x$, 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.

Our first axiom is that the injections are both imps.

$$I \sqsupseteq (\subset\!\!\rightarrow \circ \leftarrow\!\!\supset \cup) \sqcup (\subset\!\!\rightarrow \circ \leftarrow\!\!\supset \cup). \tag{5}$$

The "dual" of this axiom that we propose is:

$$I \sqsupseteq (\ll \cup \circ \ll) \sqcap (\gg \cup \circ \gg), \tag{6}$$

which says that projecting a pair onto its first and second components and then recombining the components leaves the pair unchanged.

We remark that axioms (5) and (6) take the following form when rephrased in terms of the product and sum operations.

$$I \sqsupseteq I + I \quad \text{and} \quad I \sqsupseteq I \times I. \tag{7}$$

This is reassuring since it is one step on the way to guaranteeing that $+$ and $\times$ are binary relators. Product and conjunction are closely related. Specifically, we have (in the set-theoretic interpretation of $\times$)

$$x \langle P \cap Q \rangle y \equiv (x, x) \langle P \times Q \rangle (y, y).$$

Abstracting from this property in order to find an axiom that has a pleasing syntactic shape we are led to the following axiom:

$$(P \vartriangle Q) \cup \circ (R \vartriangle S) = (P \cup \circ R) \sqcap (Q \cup \circ S). \tag{8}$$

The dual axiom for sum is

$$(P \triangledown Q) \circ (R \triangledown S) \cup = (P \circ R \cup) \sqcup (Q \circ S \cup). \tag{9}$$

As a final axiom we postulate that left projection is possible if and only if right projection is possible:

$$\ll > = \gg > \tag{10}$$

Property (10) is equivalent to $\top\top \circ \ll = \top\top \circ \gg$. Its dual is therefore the trivially true: $\hookrightarrow \circ \bot\!\!\!\bot = \hookleftarrow \circ \bot\!\!\!\bot$. There are thus no further axioms for sum.

From the axioms it is possible to prove that $\ll$, $\gg$ are imps and $\hookrightarrow$, $\hookleftarrow$ are imps and co-imps. Furthermore, we have

Theorem 2.4. $\times$ *and* $+$ *are binary relators.*

Also, we have the following distribution properties:

$$P \vartriangle Q \circ R \sqsubseteq (P \circ R) \vartriangle (Q \circ R), \tag{11}$$

$$P \vartriangle Q \circ f = (P \circ f) \vartriangle (Q \circ f), \text{ for all imps } f, \tag{12}$$

$$R \circ P \triangledown Q = (R \circ P) \triangledown (R \circ Q). \tag{13}$$

Property (12) is called split-imp fusion and property (13) is called spec-junc fusion.

Another way of characterising $\triangledown$ is given in the following theorem.

Theorem 2.5 (Unique extension property). *For all specs* R, S *and* X,

$$X \circ (I + I) = R \triangledown S \quad \equiv \quad X \circ \hookrightarrow = R \wedge X \circ \hookleftarrow = S.$$

Two direct consequences are the computation rules for $\triangledown$ and $+$:

Corollary 2.6 (Computation rules).

(a) $R \triangledown S \circ \hookrightarrow = R$.
(b) $R \triangledown S \circ \hookleftarrow = S$.
(c) $R + S \circ \hookrightarrow = \hookrightarrow \circ R$.
(d) $R + S \circ \hookleftarrow = \hookleftarrow \circ S$.

Due to the non-determinism, the unique extension property for $\vartriangle$ is conditional:

Theorem 2.7 (Unique extension property). *For all specs* R *and* S *and spec* X *such that* $I \times I \circ X = (\ll \circ X) \vartriangle (\gg \circ X)$,

$$(I \times I) \circ X = R \vartriangle S \quad \equiv \quad \ll \circ X = R \circ S > \wedge \gg \circ X = S \circ R > .$$

Note that the condition of the above theorem holds if X is an imp. Furthermore, it also holds if X is a split which gives us the computation rules for $\vartriangle$ and $\times$:

Corollary 2.8 (Computation rules).
(a) $\ll \circ R \vartriangle S = R \circ S >$.
(b) $\gg \circ R \vartriangle S = S \circ R >$.
(c) $\ll \circ R \times S = R \circ \ll \circ I \times S >$.
(d) $\gg \circ R \times S = S \circ \gg \circ R > \times I$.

2.3.2. Unit

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

Unit: $\bot\!\!\bot \neq \mathbb{1} \wedge I \sqsupseteq \mathbb{1} \circ \top\!\!\top \circ \mathbb{1}$

Two of the main properties of $\mathbb{1}$ are that it is a monotype and that it is an atom.

2.3.3. Map relators

In this section we define the so-called "map" relators via least fixed points. Before doing so we introduce the notion of a catamorphism, the generalisation of the fold operator in functional programming languages.

Given a relator F we define μF to be the least solution of $X :: X = F.X$. In general, one can view μF (which is always a monotype) as the recursively defined type corresponding to the relator F. Corresponding with μF we can define the so-called catamorphism operator.

Definition 2.9 (*Catamorphism*). Catamorphism $(\![F; R]\!)$ is defined to be the least solution of

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

Catamorphisms can be viewed as recursively defined specs which follow the same recursion pattern as the elements of μF. For catamorphisms we have another characterisation which is more suitable for calculations.

Theorem 2.10 (Unique extension property). *For relator F and specs X and R,*

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

When there is no doubt about the relator in question we will drop the argument F within the catamorphism brackets.

The coincidence in $(\![R]\!)$ of the least and greatest solutions of the right-hand side of Theorem 2.10 together with the Knaster–Tarski theorem gives the following theorem.

Theorem 2.11.
(a) $X \sqsupseteq ([R]) \Leftarrow X \sqsupseteq R \circ F.X \circ \mu F$.
(b) $X \sqsubseteq ([R]) \Leftarrow X \sqsubseteq R \circ F.X \circ \mu F$.

A corollary of the unique extension property and the above that figures very prominently in program calculations is the following corollary.

Corollary 2.12 (Catamorphism fusion).
(a) $U \circ ([V]) = ([R]) \Leftarrow U \circ V = R \circ F.U$.
(b) $U \circ ([V]) \sqsupseteq ([R]) \Leftarrow U \circ V \sqsupseteq R \circ F.U$.
(c) $U \circ ([V]) \sqsubseteq ([R]) \Leftarrow U \circ V \sqsubseteq R \circ F.U$.

In earlier publications [3,22] instead of fusion laws, the term "promotion" property was used, this term having been used by Bird to name a technique for improving the efficiency of programs [9] and which our notion captures and generalised. Maarten Fokkinga [17] suggested the more descriptive term "fusion" property, and we have been glad to adopt his suggestion.

If the relator F is of the shape $I \oplus$, i.e. $F.X = I \oplus X$ where $\oplus$ is a binary relator, we can define the so-called map operator.

Theorem 2.13 (Map). *The function $*$ from specs to specs defined by*

$$*R = ([I \oplus; R \oplus I])$$

is a relator.

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

This concludes our summary of the algebraic framework.

3. Binary structures

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

$$F.X = I + (\mathbb{1} + X \times X). \tag{14}$$

3.1. Constructors and computation rules

For the relator F defined above we have the following four constructors:

$$\tau = \mu F \circ \subset\!\!\!\rightarrow = \subset\!\!\!\rightarrow, \tag{15}$$

$$\eta = \mu F \circ \hookleftarrow = \hookleftarrow \circ \mathbb{1} + \mu F \times \mu F, \tag{16}$$

$$\square = \mu F \circ \hookleftarrow \circ \subset\!\!\!\rightarrow = \hookleftarrow \circ \subset\!\!\!\rightarrow \circ \mathbb{1}, \tag{17}$$

$$\# = \mu F \circ \hookleftarrow \circ \hookleftarrow = \hookleftarrow \circ \hookleftarrow \circ \mu F \times \mu F. \tag{18}$$

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

$$\eta = \square \triangledown \#.$$

So, η is just the combination of two other constructors. Elements of μF we call bins (short for binary structures); bins correspond to Meertens' D-structures.

We may assume without loss of generality that every F-catamorphism, where F is given by (14) 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 3.1 (Computation rules).

(a) $(\!|R \triangledown (S \triangledown \otimes)|\!) \circ \tau = R$.

(b) $(\!|R \triangledown (S \triangledown \otimes)|\!) \circ \square = S \circ \mathbb{1}$.

(c) $(\!|R \triangledown (S \triangledown \otimes)|\!) \circ \# = \otimes \circ (\!|R \triangledown (S \triangledown \otimes)|\!) \times (\!|R \triangledown (S \triangledown \otimes)|\!)$.

Also, we can state the unique extension property using the constructors, as in the following theorem.

Theorem 3.2. *For spec X such that $X = X \circ \mu F$,*

$$\begin{aligned} & X = (\!|R \triangledown (S \triangledown \otimes)|\!) \\ \equiv\ & \\ & X \circ \tau = R \wedge X \circ \square = S \circ \mathbb{1} \wedge X \circ \# = \otimes \circ X \times X. \end{aligned}$$

For later use we define the following shorthand:

Definition 3.3 (*Empty tree*). $\varepsilon = \square \circ \top\top$.

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

Definition 3.4 (*Point*). Spec x is a *point* iff x is an imp $\wedge$ $x = x \circ \top\top$.

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

We introduce the following abbreviations:

$$R \otimes S = \otimes \circ R \vartriangle S, \tag{19}$$

$$R \underline{\otimes} S = (R \circ \ll) \otimes (S \circ \gg) = \otimes \circ R \times S, \tag{20}$$

For $R \otimes S$ we have by split-imp fusion (12):

$$R \otimes S \circ f = (R \circ f) \otimes (S \circ f), \quad \text{for imp } f. \tag{21}$$

Note that the computation rule for $\#$ can now be rewritten as:

$$([R \triangledown (S \triangledown \otimes)]) \circ \# = ([R \triangledown (S \triangledown \otimes)]) \underline{\otimes} ([R \triangledown (S \triangledown \otimes)]). \tag{22}$$

3.2. *Map and reduce*

In this section we define the map and reduce operators. Instantiating the definition of map, Theorem 2.13, gives us for the relator F for the binary structures the following theorem.

Theorem 3.5 (Map). *The function $*$ from specs to specs defined by*

$$*R = ([R + (\mathbb{1} + I \times I)])$$

is a relator.

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 3.6 (Map fusion). $([P \triangledown Q]) \circ *R = ([(P \circ R) \triangledown Q])$.

Theorem 3.7 (Map – alternative definition). $*R = ([(\tau \circ R) \triangledown \eta])$.

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

Instantiating the computation rule, Theorem 3.1, with the above definition of $*$ we obtain the following computation rules:

$$*R \circ \tau = \tau \circ R,$$

$$*R \circ \square = \square,$$

$$*R \circ \# = \# \circ *R \times *R.$$

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

Another 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 remain 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 [8] 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 arguments 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. That is for all specs R and S,

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

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

$$/S \circ *R$$

$=$ {We try to express $/S$ as a catamorphism
• $/S = (\!|P \triangledown Q|\!)$}

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

$=$ {map fusion: (Theorem 3.6}

$$(\!|(P \circ R) \triangledown Q|\!)$$

$=$ {choose $P = I$ and $Q = S$}

$$(\!|R \triangledown S|\!).$$

Thus, if we define the reduce operator by

$$/S = ([I \triangledown S]), \tag{23}$$

then we have established the following factorisation property.

Lemma 3.8 $/S \circ *R = ([R \triangledown S])$.

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 3.9 ($/\eta$ leapfrog).
(a) $/S \circ /\eta = /S \circ */S$.
(b) $*R \circ /\eta = /\eta \circ **R$.

3.3. 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. [20].

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 3.10 (*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 $p> \sqcap (\neg p)> = \bot\bot$. 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 also sometimes write *true* instead of $\top\top$ and *false* instead of $\bot\bot$. We define "conditional" now.

Definition 3.11 (*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 [19] 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 [7] an alternative definition of conditionals that eliminates the need to restrict p to a right condition is proposed. Moreover, a more comprehensive account of their properties is included.

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 [20]. The subset of those properties that we use in the present paper is given in the following theorem.

Theorem 3.12. *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) $R \lhd p \rhd S \circ f = (R \circ f) \lhd (p \circ f) \rhd (p \circ f)$.

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

The last two properties we conclude this section with, are given in the following theorem.

Theorem 3.13 ($\_ \otimes \_$-Conditional abides law).

(a) $(R \lhd p \rhd S) \otimes (P \lhd p \rhd Q) = (R \otimes P) \lhd p \rhd (S \otimes Q)$.
(b) $(R \lhd p \rhd S) \otimes (P \lhd q \rhd Q) = ((R \otimes P) \lhd q \rhd (R \otimes Q)) \lhd p \rhd ((S \otimes P) \lhd q \rhd (S \otimes Q))$.

3.4. Filters

The definition of filter is borrowed directly from the work of Meertens [23] and Bird [14]:

Definition 3.14 (*Filter*). For right-condition p,

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

Note that from the fact that τ and ε are imps and the fact that conditionals, junc and catamorphisms respect imps it follows that $\lhd p$ 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 6.

By design $\lhd true$ is the identity function on specs of the correct type.

Theorem 3.15. $\lhd true = *I$.

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

Lemma 3.16.

(a) $*R \circ \lhd p = /\eta \circ *((\tau \circ R) \lhd p \rhd \varepsilon)$.

(b) $/\eta \circ *R \circ \lhd p = /\eta \circ *(R \lhd p \rhd \varepsilon)$.

A direct consequence of Lemma 3.16(b) combined with Theorem 3.12(f) is the following theorem.

Theorem 3.17 ($\lhd$ distribution). $\lhd p \circ \lhd q = \lhd(p \wedge q)$.

Yet another fusion property for filters is given in the following theorem.

Theorem 3.18 (Filter range translation). *For all imps* f,

$$\lhd p \circ *f = *f \circ \lhd(p \circ f) \circ *f> .$$

4. Laws

Map relators (of which the bin relator is an instance) define types free from laws. The type structures within the Boom-hierarchy, however, are characterised by laws relating differing binary structures to each other. We discuss these laws in the next section. In the current section we say what it is for a binary spec to be symmetric,

associative, idempotent and to have left and right units. So-called "sectioning" is introduced here in order to define units, but it will also prove useful later in other contexts.

4.1. Sections

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 generalise it to a left-condition because left-conditions can be viewed as the relational generalisation of constants. We define left-sectioning as follows.

Definition 4.1 (*Left-sections*). For p a left-condition, and $\otimes$ a binary spec, we define

$$p\otimes = p \otimes I.$$

(See (19) for the definition of $p \otimes I$).

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

Theorem 4.2 (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 Theorem 4.2 holds for arbitrary spec R. Similarly, we define right-sectioning as follows.

Definition 4.3 (*Right-sections*). For left condition q, and binary spec $\otimes$,

$$\otimes q = I \otimes q.$$

We have the following property for right-sections.

Theorem 4.4 (Right-sectioning). $\otimes q \circ R = R \otimes q$.

4.2. Symmetry, associativity and idempotence

As usual, we say that $\otimes$ is symmetric if $R \otimes S = S \otimes R$ for all specs R and S. There is a dummy-free characterization of symmetry:

Theorem 4.5 (Symmetry). $\otimes$ *is symmetric iff* $\otimes = \otimes \circ \gamma$ *where* $\gamma = \gg \Delta \ll$ i.e. γ *is a natural transformation such that* $R \times S \circ \gamma = \gamma \circ S \times R$ *for all specs* R *and* S.

Similarly, we say that $\otimes$ is associative if $R \otimes (S \otimes T) = (R \otimes S) \otimes T$ for all specs R, S and T. Again, there is a dummy-free characterisation of associativity:

Theorem 4.6 (Associativity). $\otimes$ *is associative iff* $\otimes \circ I \times \otimes = \otimes \circ \otimes \times I \circ \beta$ *where* $\beta = (I \times \ll) \triangle (\gg \circ \gg)$, i.e. β *is a natural transformation such that* $(R \times S) \times T \circ \beta = \beta \circ R \times (S \times T)$ *for all specs* R, S *and* T.

For idempotency we must be careful. Were we to define idempotency by $R \otimes R = R$ for all specs R, we would have in particular $I \otimes I = I$. Since $I \otimes I = \otimes \circ I \triangle I$ this implies the very strong surjectivity condition $\otimes< = I$. Fortunately, a simple definition is at hand.

Definition 4.7 (*Idempotence*). $\otimes$ is idempotent iff $I \otimes I \sqsubseteq I$.

4.3. Unit

We define left-units by

Definition 4.8 (*Left-unit*). $l_\otimes$ is a left-unit of $\otimes$ iff

$$left\_condition.l_\otimes \wedge (\otimes \circ \ll \cup)> \sqsupseteq l_\otimes< \wedge\ l_\otimes \otimes = (\otimes \circ \gg \cup)> .$$

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

Of course, we can give a dual definition for $r_\otimes$ being a right-unit of $\otimes$.

Definition 4.9 (*Right-unit*). $r_\otimes$ is a right-unit of $\otimes$ iff

$$left\_condition.r_\otimes \wedge (\otimes \circ \gg \cup)> \sqsupseteq r_\otimes< \wedge\ \otimes r_\otimes = (\otimes \circ \ll \cup)> .$$

For an 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.

$r_\otimes$

$=$ {definition left-, right-unit: $I \sqsupseteq l_\otimes \otimes \sqsupseteq r_\otimes<$}

$l_\otimes \otimes \circ r_\otimes$

$=$ {sectioning}

$\otimes r_\otimes \circ l_\otimes$

$=$ {definition left-, right-unit: $I \sqsupseteq \otimes r_\otimes \sqsupseteq l_\otimes<$}

$l_\otimes$.

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

Definition 4.10 (*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 ourselves to write just $/\otimes$ instead of the cumbersome $/(1_\otimes \triangledown \otimes)$.

5. 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 $\varepsilon \# x$ represent the same tree. Also, we can represent a list by a tree if we forget about the order in which sublists are joined, i.e. $x \# (y \# z)$ and $(x \# y) \# z$ represent the same list. A bag can be represented by a list if we forget about the ordering of the elements: $x \# y$ and $y \# x$ 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: $x \# x$ and x represent the same set. We now give a construction for a congruence relation on μF such that the equivalence classes of that congruence relation are just those bins (trees, lists, bags) that represent the same tree (list, bag, set). We denote the four congruence relations by *Tree*, *List*, *Bag* and *Set*.

5.1. Congruence relations

By definition, α is a congruence relation on μF if it is an equivalence relation (*er*) on μF and it is F-substitutive. That is, α is symmetric and transitive

$$\alpha = \alpha\cup \;\wedge\; \alpha \sqsupseteq \alpha \circ \alpha, \tag{24}$$

it is reflexive on μF,

$$\alpha \sqcap I = \mu F,$$

or equivalently,

$$\alpha \sqsupseteq \mu F \;\wedge\; \alpha \circ \mu F = \alpha, \tag{25}$$

and it is F-substitutive:

$$\alpha \sqsupseteq F.\alpha. \tag{26}$$

Conditions (24)–(26) define an F-congruence whatever the relator F, not just the particular one we are studying here. However, for our particular needs it pays to rephrase (26) in terms of the constructor $\#$. This is achieved by noting that for all *er*s X and Y

$$X \sqsupseteq Y \equiv X = X \circ Y \quad (= Y \circ X). \tag{27}$$

Note that relator F maps *er*s on μF to *er*s on μF. Hence, (26) is equivalent to

$$\alpha \circ F.\alpha = \alpha .$$

Now,

$$\begin{aligned} & \alpha \circ F.\alpha \\ = \ & \{\text{definition of } F, \tau, \Box \text{ and } \# \} \\ & \alpha \circ \tau \nabla((\Box \nabla(\# \circ \alpha \times \alpha)) \\ = \ & \{\text{spec-junc fusion}\} \\ & (\alpha \circ \tau)\nabla((\alpha \circ \Box)\nabla(\alpha \circ \# \circ \alpha \times \alpha)). \end{aligned}$$

Similarly,

$$\alpha = \alpha \circ \mu F = (\alpha \circ \tau)\nabla((\alpha \circ \Box)\nabla(\alpha \circ \#)).$$

Thus, by the unique extension property of ∇, for all *er*s on μF,

$$\alpha \text{ is } F\text{-substitutive } \equiv \alpha = \alpha \circ \# \circ \alpha \times \alpha. \tag{28}$$

The right-hand side of (28) we call the *absorption rule.*

5.2. *Tree, List, Bag and Set*

Next we give a formal definition of the four congruence relations *Tree*, *List*, *Bag* and *Set*. Let $\mathscr{L}$ be one of these relations; we define the following constructors.

$$\tau_{\mathscr{L}} = \mathscr{L} \circ \tau . \tag{29}$$

$$\Box_{\mathscr{L}} = \mathscr{L} \circ \Box . \tag{30}$$

$$\#_{\mathscr{L}} = \mathscr{L} \circ \# . \tag{31}$$

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.

Definition 5.1 (*Boom-hierarchy*). *Tree* is the least congruence relation such that ε_{Tree} is the unit of $\#_{Tree}$:

(a) $\#_{Tree} \circ \varepsilon_{Tree} \Delta Tree = Tree.$

(b) $\#_{Tree} \circ Tree \Delta \varepsilon_{Tree} = Tree.$

List is the least congruence relation such that $List \supseteq Tree$ and $\#_{List}$ is associative:

(c) $\#_{List} \circ \#_{List} \times I = \#_{List} \circ I \times \#_{List} \circ \beta.$

Bag is the least congruence relation such that $Bag \supseteq List$ and $\#_{Bag}$ is symmetric:

(d) $\#_{Bag} = \#_{Bag} \circ \gamma.$

Set is the least congruence relation such that $Set \supseteq Bag$ and $\#_{Set}$ is idempotent:

(e) $\#_{Set} \circ I \Delta I = Set.$

(β and γ are the natural transformations as used for the definition of associativity, Theorem 4.6, and symmetry, Theorem 4.5.)

An obvious question is now: is this a good definition? That it is is well known, but it may be helpful to outline the argument.

The basic idea is to appeal to the Knaster–Tarski theorem: to do so we transform the requirements for *Tree*, *List*, *Bag* and *Set* into a fixpoint equation in the complete lattice of *er*s on μF.

By unfolding the definition of $\#_{\mathscr{L}}$ and $\varepsilon_{\mathscr{L}}$ and using the absorption rule, we can shift the type information in (a)–(e) entirely to the left, which gives us:

(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) $Set \circ \# \circ I \triangle I = Set$.

The requirements (a)–(e) all have the same form, namely

$$\mathscr{L} \circ f = \mathscr{L} \circ g,$$

where $f> = g>$ and imp.f and imp.g.

(In the case of (a), (b) and (e), you have to realise that $\mathscr{L} = \mathscr{L} \circ \mu F$, and instantiate g to μF.) Now using properties of imps and of *er*s we can prove that for all *er*s X and all f and g satisfying the above conditions

$$X \circ f = X \circ g \equiv X \sqsupseteq f \circ g\cup.$$

So, denoting by $f_a \ldots f_e$ and $g_a \ldots g_e$ the pairs of imps occurring in requirements (a)–(e) we have, for example, that *Tree* is the least congruence relation containing $f_a \circ g_a\cup$ and $f_b \circ g_b\cup$. Thus, *Tree* is the least solution of

$$X : er.X:\ X \sqsupseteq (f_a \circ g_a\cup) \sqcup (f_b \circ g_b\cup) \sqcup F.X.$$

But for *er* X and spec R,

$$X \sqsupseteq R \equiv X \sqsupseteq (R \sqcup R\cup)^*,$$

where S^* denotes the least transitive and reflexive-on-μF spec containing S. Thus, *Tree* is the least solution of the equation:

$$X : er.X:\ X \sqsupseteq (\mathscr{E}_a \sqcup \mathscr{E}_b \sqcup F.X)^*,$$

where $\mathscr{E}_a = f_a \circ g_a\cup \sqcup g_a \circ f_a\cup$ and $\mathscr{E}_b = f_b \circ g_b\cup \sqcup g_b \circ f_b\cup$.

Since the right side of this equation is a monotonic endofunction on *er* X, and the *er*s form a complete lattice, it follows by Knaster–Tarski that the equation has a least solution.

Similarly, we can transform the definitions of *List*, *Bag* and *Set* into a least fixpoint equation. Hence, Definition 5.1 is a valid definition. For more details the reader is referred to [27] where a more general construction is given using *partial* equivalence

relations instead of *ers* on μF and which permits the combination of laws *and* restrictions on types.

By definition, we have the following lattice ordering:

$$\mu F \sqsubseteq Tree \sqsubseteq List \sqsubseteq Bag \sqsubseteq Set . \tag{32}$$

Using property (27) it follows that

$$Tree = Tree \circ \mu F , \tag{33}$$

$$List = List \circ Tree = List \circ \mu F , \tag{34}$$

$$Bag = Bag \circ List = Bag \circ Tree = Bag \circ \mu F , \tag{35}$$

$$Set = Set \circ Bag = Set \circ List = Set \circ Tree = Set \circ \mu F . \tag{36}$$

5.3. Respecting laws

Definition 5.2. We say that spec R *respects* an equivalence relation $\mathscr{L}$ iff

$$R \circ \mathscr{L} = R .$$

Informally, if spec R respects $\mathscr{L}$ it means that spec R does not differentiate between representatives of the same equivalence class of $\mathscr{L}$. For instance, if $R \circ Set = R$ holds then spec R yields the same result when applied to bins x and $x \uplus x$. Furthermore, from equations (33)–(36) it follows that if a spec respects a type from the Boom-hierarchy, it also respects a type lower in the level of the Boom-hierarchy. For instance, if a spec R respects *Set*, it also respects *List*.

For each level of the Boom-hierarchy we have the following conditions for a catamorphism to respect the corresponding equivalence relation.

Theorem 5.3 (Type of catamorphism). *In order to state the theorem let us introduce the following abbreviations. First let* $\mathscr{C}$ *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:*

(a) $(S \circ \top\top) \underline{\otimes} \circ \mathscr{C} = \mathscr{C}$,

(b) $\underline{\otimes} (S \circ \top\top) \circ \mathscr{C} = \mathscr{C}$,

(c) $(\mathscr{C} \underline{\otimes} \mathscr{C}) \otimes \mathscr{C} = \mathscr{C} \underline{\otimes} (\mathscr{C} \underline{\otimes} \mathscr{C}) \circ \beta$,

(d) $\mathscr{C} \underline{\otimes} \mathscr{C} = \mathscr{C} \underline{\otimes} \mathscr{C} \circ \gamma$,

(e) $\mathscr{C} \otimes \mathscr{C} = \mathscr{C}$.

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

Now we can state the theorem, which is:

$\mathscr{C}$ *respects Tree* $\equiv$ (a) $\wedge$ (b).

$\mathscr{C}$ *respects List* $\equiv$ $\mathscr{C}$ *respects Tree* $\wedge$ (c).

$\mathscr{C}$ *respects Bag* $\equiv$ $\mathscr{C}$ *respects List* $\wedge$ (d).

$\mathscr{C}$ *respects Set* $\equiv$ $\mathscr{C}$ *respects Bag* $\wedge$ (e).

Next 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). The following is an important theorem.

Theorem 5.4 (Type-η fusion). *For $\mathscr{L}$ equal to Tree, List, Bag or Set,*

$$\mathscr{L} \circ ([\![R \triangledown \eta]\!]) = ([\![(\mathscr{L} \circ R) \triangledown (\mathscr{L} \circ \eta)]\!]).$$

Using this property we can prove the following type judgements.

Theorem 5.5. *For $\mathscr{L}$ equal to Tree, List or Bag, and imp f,*

(a) $\mathscr{L} \circ ([\![R \triangledown \eta]\!])$ *respects* $\mathscr{L}$,
(b) $Set \circ ([\![f \triangledown \eta]\!])$ *respects Set*,
(c) $\mathscr{L} \circ *R$ *respects* $\mathscr{L}$,
(d) $Set \circ *f$ *respects Set*,
(e) $\mathscr{L} \circ \lhd p$ *respects* $\mathscr{L}$,
(f) $Set \circ \lhd p$ *respects Set*.

The importance of Theorem 5.5 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 $R \circ \lhd p$ also respects *Bag*. Thus, using these theorems we can move around the type information in our formulae.

6. Comparison with quantifier notation

In this section we make a link to the quantifier notation. We define

$$\otimes (i : p.i : R.i) = /\otimes \circ *R \circ \lhd p, \tag{37}$$

where $R.i$ is defined by

$$R.i = R \circ i\bullet, \tag{38}$$

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 $\mathscr{A}$ is an enumeration of the elements of the type A and $\otimes$ is both associative and symmetric then we make the definition

$$\otimes (i \in A : p.i : R.i) = /\otimes \circ *R \circ \lhd p \circ \mathscr{A}. \tag{39}$$

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.

6.1 Trading and translation

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 Lemma 3.16(a) and Theorem 3.9 it is easy to prove that

$$/\otimes \circ *R \circ \lhd p = /\otimes \circ *(R \lhd p \rhd 1_{\otimes}). \tag{40}$$

The well-known trading rules for universal and existential quantification are immediate corollaries of (40).

$$/\mathbf{and} \circ *\bar{q} \circ \lhd p = /\mathbf{and} \circ *(\overline{p \Rightarrow q}), \tag{41}$$

$$/\mathbf{or} \circ *\bar{q} \circ \lhd p = /\mathbf{or} \circ *(\overline{p \wedge q}), \tag{42}$$

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

$$\bar{q} = \mathbf{true} \lhd q \rhd \mathbf{false},$$

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 (41) and (42) it suffices to observe that

$$\bar{q} \lhd p \rhd \mathbf{true} = \overline{p \Rightarrow q} \tag{43}$$

and

$$\bar{q} \lhd p \rhd \mathbf{false} = \overline{p \wedge q}, \tag{44}$$

which follows directly from the definition of the bar and the properties of conditionals. As an example we prove (43)

$\bar{q} \lhd p \rhd \mathbf{true}$

$=$ {property conditionals: Theorem 3.12(d)}

$\mathbf{true} \lhd \neg p \rhd \bar{q}$

$=$ {definition bar}

$\mathbf{true} \lhd \neg p \rhd (\mathbf{true} \lhd q \rhd \mathbf{false})$

$=$ {property conditionals: Theorem 3.12(g)}

$\mathbf{true} \lhd \neg p \vee q \rhd \mathbf{false}$

$=$ {definition $\Rightarrow$, definition bar}

$\overline{p \Rightarrow q}$.

Less familiar consequences of (40) are the trading rules for equivalence and inequivalence.

$$/\equiv \circ *\bar{q} \circ \lhd p = /\equiv \circ *(\overline{p \Rightarrow q}), \tag{45}$$

$$/\not\equiv \circ *\bar{q} \circ \lhd p = /\not\equiv \circ *(\overline{p \wedge q}). \tag{46}$$

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

The literal translation of (41) and (42) 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 Lemma 3.17:

$$\lhd(p \wedge r) = \lhd p \circ \lhd r. \tag{47}$$

The second rule we have already derived is range translation. From the filter range translation rule, Theorem 3.18, and compositionality of map it follows that, for imp f,

$$/\otimes \circ *R \circ \lhd p \circ *f = /\otimes \circ *(R \circ f) \circ \lhd(p \circ f) \circ *f> . \tag{48}$$

In the quantifier notation this would be expressed as

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

In the following sections we derive 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.

6.2. Unit law

In this section we concentrate on trees. Using the results of the previous section and the property of the constructors, Definition 5.1(a), we can prove the unit rule:

Theorem 6.1 (Unit). $Tree \circ \lhd false = Tree \circ \varepsilon \circ \mu F$.

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

Using Theorem 6.1 we can prove the following corollary.

Corollary 6.2 (Unit rule). *If* $([R \triangledown (S \triangledown \otimes)])$ *respects Tree then*

$$([R \triangledown (S \triangledown \otimes)]) \circ \lhd false = S \circ \top\top \circ \mu F .$$

Proof.

$$\begin{aligned}
& ([R \triangledown (S \triangledown \otimes)]) \circ \lhd false \\
= \ & \{([R \triangledown (S \triangledown \otimes)]) \text{ respects } Tree\} \\
& ([R \triangledown (S \triangledown \otimes)]) \circ Tree \circ \lhd false \\
= \ & \{\text{Theorem 6.1}\} \\
& ([R \triangledown (S \triangledown \otimes)]) \circ Tree \circ \varepsilon \circ \mu F \\
= \ & \{([R \triangledown (S \triangledown \otimes)]) \text{ respects } Tree\} \\
& ([R \triangledown (S \triangledown \otimes)]) \circ \varepsilon \circ \mu F \\
= \ & \{\text{definition } \varepsilon\} \\
& ([R \triangledown (S \triangledown \otimes)]) \circ \Box \circ \top\top \circ \mu F \\
= \ & \{\text{computation rule } \Box\} \\
& S \circ \top\top \circ \mu F . \qquad \Box
\end{aligned}$$

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

$$/+ \circ * R \circ \lhd false = 0\bullet \circ \mu F ,$$

$$/\times \circ * R \circ \lhd false = 1\bullet \circ \mu F ,$$

$$/\mathbf{and} \circ * R \circ \lhd false = \mathbf{true} \circ \mu F ,$$

$$/\mathbf{or} \circ * R \circ \lhd false = \mathbf{false} \circ \mu F ,$$

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:

$$\Sigma(i : false : R.i) = 0,$$

$$\Pi(i : false : R.i) = 1,$$

$$\forall(i : false : R.i) = true,$$

$$\exists(i : false : R.i) = false.$$

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 not lists or trees.

6.3. *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 6.3 ($\underline{\otimes}$-$\otimes$ abides law).

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

Proof.

$$\begin{aligned}
&(R \mathbin{\underline{\otimes}} S) \otimes (P \mathbin{\underline{\otimes}} Q) \\
= &\ \{\text{property (20)}\} \\
&((R \circ \ll) \otimes (S \circ \gg)) \otimes ((P \circ \ll) \otimes (Q \circ \gg)) \\
= &\ \{\otimes \text{ associative and symmetric}\} \\
&((R \circ \ll) \otimes (P \circ \ll)) \otimes ((S \circ \gg) \otimes (Q \circ \gg)) \\
= &\ \{\ll, \gg \text{ imps, split-imp fusion (21)}\} \\
&(R \otimes P \circ \ll) \otimes (S \otimes Q \circ \gg) \\
= &\ \{\text{property (20)}\} \\
&(R \otimes P) \mathbin{\underline{\otimes}} (S \otimes Q). \qquad \square
\end{aligned}$$

By applying the unique extension property, Theorem 3.2, we also have the following theorem.

Theorem 6.4 (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, the following theorem.

Theorem 6.5.

$$([\![(R \otimes S)\triangledown(U\triangledown\otimes)]\!]) \circ \triangleleft p$$
$$=$$
$$(([\![R\triangledown(U\triangledown\otimes)]\!]) \circ \triangleleft p) \otimes (([\![S\triangledown(U\triangledown\otimes)]\!]) \circ \triangleleft p).$$

Proof.

$$([\![(R \otimes S)\triangledown(U\triangledown\otimes)]\!]) \circ \triangleleft p$$
$$= \{\text{asso-sym rule: Theorem 6.4}\}$$
$$([\![R\triangledown(U\triangledown\otimes)]\!]) \otimes ([\![S\triangledown(U\triangledown\otimes)]\!]) \circ \triangleleft p$$
$$= \{\triangleleft p \text{ imp, split-imp fusion (21)}\}$$
$$(([\![R\triangledown(U\triangledown\otimes)]\!]) \circ \triangleleft p) \otimes (([\![S\triangledown(U\triangledown\otimes)]\!]) \circ \triangleleft p). \qquad \square$$

Theorem 6.5 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 6.4 holds with $U := Bag \circ \square$ and $\otimes := Bag \circ \#$. This results in the following theorem.

Theorem 6.6 (Asso-sym rule).

$$Bag \circ ([\![(R \# S)\triangledown\eta]\!]) = Bag \circ ([\![R\triangledown\eta]\!]) \# ([\![S\triangledown\eta]\!]).$$

For the proof of range splitting (below) and range disjunction (in the next section) we will use Theorem 6.6 to move the join operator inside the catamorphism, then we use Theorem 3.13(a) or (b) 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 6.7 (Range splitting). $Bag \circ \triangleleft p \# \triangleleft \neg p = Bag$.

Proof. First we prove:

$$
\begin{aligned}
& Bag \circ (\tau \lhd p \rhd \varepsilon) \mathbin{\#} (\tau \lhd \neg p \rhd \varepsilon) \\
= \ & \{\text{property conditionals: Theorem 3.12(d)}\} \\
& Bag \circ (\tau \lhd p \rhd \varepsilon) \mathbin{\#} (\varepsilon \lhd p \rhd \tau) \\
= \ & \{\mathbin{\#} \text{ abides with } \lhd p \rhd\text{: Theorem 3.13(a)}\} \\
& Bag \circ (\tau \mathbin{\#} \varepsilon) \lhd p \rhd (\varepsilon \mathbin{\#} \tau) \\
= \ & \{\text{property conditionals: Theorem 3.12(j)}\} \\
& (Bag \circ \tau \mathbin{\#} \varepsilon) \lhd p \rhd (Bag \circ \varepsilon \mathbin{\#} \tau) \\
= \ & \{\text{property constructors: Definition 5.1(a) and (b)}\} \\
& (Bag \circ \tau) \lhd p \rhd (Bag \circ \tau) \\
= \ & \{\text{property conditionals: Theorem 3.12(a)}\} \\
& Bag \circ \tau .
\end{aligned}
$$

And from this the theorem follows:

$$
\begin{aligned}
& Bag \circ \lhd p \mathbin{\#} \lhd \neg p \\
= \ & \{\text{definition filter}\} \\
& Bag \circ (\llbracket (\tau \lhd p \rhd \varepsilon) \triangledown \eta \rrbracket) \mathbin{\#} (\llbracket (\tau \lhd \neg p \rhd \varepsilon) \triangledown \eta \rrbracket) \\
= \ & \{\text{asso-sym rule: Theorem 6.6}\} \\
& Bag \circ (\llbracket ((\tau \lhd p \rhd \varepsilon) \mathbin{\#} (\tau \lhd \neg p \rhd \varepsilon)) \triangledown \eta \rrbracket) \\
= \ & \{\text{type-}\eta \text{ fusion: Theorem 5.4}\} \\
& (\llbracket (Bag \circ (\tau \lhd p \rhd \varepsilon) \mathbin{\#} (\tau \lhd \neg p \rhd \varepsilon)) \triangledown (Bag \circ \eta) \rrbracket) \\
= \ & \{\text{calculation above}\} \\
& (\llbracket (Bag \circ \tau) \triangledown (Bag \circ \eta) \rrbracket) \\
= \ & \{\text{type-}\eta \text{ fusion: Theorem 5.4}\} \\
& \mathrm{Bag} \circ (\llbracket \tau \triangledown \eta \rrbracket) \\
= \ & \{(\llbracket \tau \triangledown \eta \rrbracket) = \mu F,\ Bag \circ \mu F = Bag\} \\
& Bag . \qquad \square
\end{aligned}
$$

Using the range splitting rule, we can prove the following theorem.

Theorem 6.8. *If* $([R \triangledown (S \triangledown \otimes)])$ *respects Bag then*

$$([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft q$$
$$=$$
$$(([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft(p \wedge q)) \otimes (([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft(\neg p \wedge q)).$$

Proof.

$$([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft q$$
$$= \{([R \triangledown (S \triangledown \otimes)]) \text{ respects } Bag, \text{ range splitting}\}$$
$$([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft p \mathbin{\#} \triangleleft \neg p \circ \triangleleft q$$
$$= \{\text{definition } \_ \# \_, \text{ computation rule } \#\}$$
$$(([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft p) \otimes (([R \triangledown S \triangledown \otimes]) \circ \triangleleft \neg p) \circ \triangleleft q$$
$$= \{\triangleleft q \text{ imp, split-imp fusion}\}$$
$$(([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft p \circ \triangleleft q) \otimes (([R \triangledown S \triangledown \otimes]) \circ \triangleleft \neg p \circ \triangleleft q)$$
$$= \{\text{filter-distribution}\}$$
$$(([R \triangledown (S \triangledown \otimes)]) \circ \triangleleft(p \wedge q)) \otimes (([R \triangledown S \triangledown \otimes]) \circ \triangleleft(\neg p \wedge q)). \qquad \square$$

Expressed in the quantifier notation, Theorem 6.8 takes the form:

$$\otimes(i : q.i : f.i)$$
$$=$$
$$\otimes(i : p.i \wedge q.i : f.i) \otimes \otimes(i : \neg p.i \wedge q.i : f.i).$$

6.4. Extensionality

In the following section we introduce the so-called cross-product. The cross-product is not a catamorphism; it is a so-called *parameterised* catamorphism. Having a notion of extensionality one can define a parameterised catamorphism using ordinary catamorphisms. Although there is some research [6] going on towards defining other richer (recursive) structures than catamorphisms inside the relational calculus, i.e. without using points, these results are too preliminary to mention in the current paper.

By way of preparation we introduce the concept of extensionality. The extensionality axiom enables us to give point-wise proofs. Not that such proofs are preferable. On the contrary: we prefer to avoid point-wise proofs as much as possible. However, there are occasions when the avoidance of dummies becomes tortuous and point-free calculations become ugly. We postulate:

Axiom 6.9 (Extensionality).

$$R = S \equiv \forall(x : point.x : R \circ x = S \circ x).$$

Remark. From now on lower case letters x, y, z etc. denote points. Furthermore, we will drop the phrase: for all points x. So, if we say that $P(x)$ holds, we actually mean $\forall(x : point.x : P(x))$. Extensionality will thus be used as follows:

$$\begin{array}{ll} & R \circ x = S \circ x \\ \equiv & \{\text{extensionality}\} \\ & R = S. \end{array}$$

In [24] a full discussion about extensionality can be found.

For binary specs we have the following instantiation of the extensionality axiom.

Theorem 6.10 (Extensionality for binary specs). *For binary specs* $\otimes$ *and* $\oplus$,

$$\otimes = \oplus \equiv \forall(x, y :: \otimes \circ x \vartriangle y = \oplus \circ x \vartriangle y)$$

or equivalently,

$$\otimes = \oplus \equiv \forall(x, y :: x \otimes y = x \oplus y).$$

We can use extensionality to define new specs and to prove the uniqueness of such a spec. In general, we want to construct a spec R, not containing x, such that

$$R \circ x = Exp(x), \tag{49}$$

where $Exp(x)$ denotes an arbitrary expression containing x, i.e. Exp is a function from specs to specs.

Note that this is the normal way to define functions, i.e. we can define for instance *square* to be the function such that $square.x = x \times x$.

So, in the relational setting, a logical question is: for which $Exp(x)$ is it possible to construct such a spec R? Note that the left-hand side of (49) is a left-condition, so $Exp(x)$ must be a left-condition for every point x. Thus $Exp(x)$ has to satisfy

$$Exp(x) = Exp(x) \circ \top\top. \tag{50}$$

It turns out that this condition is enough in order to construct a spec R satisfying equation (49). More precisely, we have the following theorem.

Theorem 6.11 (Unique extension property). *If* $Exp(x)$ *is a left-condition for every point* x *then*

$$\begin{array}{ll} & R = \sqcup(x : point.x : Exp(x) \circ x\cup) \\ \equiv & \\ & \forall(x : point.x : R \circ x = Exp(x)). \end{array}$$

6.5. Cross-product

In this section we introduce the so-called cross-product. Conventionally, the cross-product ($\times$) is a binary spec which takes two lists and returns the list containing all pairs of the elements of both lists, for example

$$[a,b] \times [c,d,e] = [(a \times c),(b \times c),(a \times d),(b \times d),(a \times e),(b \times e)].$$

Note that in the result list the values of the left argument of the pairs run faster.

Definition 6.12 (Cross-product). We define

$$x \times = (\!|x \odot \triangledown \eta|\!),$$

where $\odot$ is defined to be the binary spec such that $\odot a = *(I \vartriangle a)$. Furthermore, we define $\times_{\otimes}$ by $\times_{\otimes} = * \otimes \circ \times$.

At the level of lists $\times_{\otimes}$ returns the list of all values $a \otimes b$ with a taken from the first list and b taken from the second list.

Note that $\times > \sqsubseteq \mu F \times \mu F$.

In this section, we adopt the convention that x, y and z denote points representing elements of μF, i.e. $\mu F \ni x<, y<$, and a denotes an arbitrary point (thus not necessarily an element of μF).

For the cross-product we have the following computation rules:

Theorem 6.13 (Cross-product computation).
(a) $x \times \circ \tau = x \odot$.
(b) $x \times \circ \Box = \Box$.
(c) $x \times \circ \# = \# \circ x \times \times x \times$.

In [13], Bird restricts discussion to lists. Here we also have

$$List \circ x \times \circ \# = List \circ \# \circ (x \times) \times (x \times)$$

and filling in the arguments we have by extensionality, for all y and z

$$List \circ x \times (y \# z) = List \circ (x \times y) \# (x \times z).$$

Thus, this definition does indeed agree with the definition of the cross-product given by Bird.

Operationally, because the elements of the first arguments of $\times$ run faster, the leapfrog property with respect to the second argument is weaker. Only when we consider bags, i.e. if we add associativity and symmetry, do we have symmetric computation rules for the right-sectioning of $\times$.

Theorem 6.14 (Cross-product computation).

(a) $\bigtimes y \circ \tau = {*}\gamma \circ y\odot$.

(b) $Tree \circ \bigtimes y \circ \Box = Tree \circ \Box$.

(c) $Bag \circ \bigtimes y \circ \# = Bag \circ \# \circ (\bigtimes y) \times (\bigtimes y)$.

(*Recall that* $\gamma = \gg_{\Delta} \ll$ *is a natural transformation such that* $R \times S \circ \gamma = \gamma \circ S \times R$.)

Proof. As an example we give the proof of part (c). It is easily proved, with the aid of extensionality, that $y\odot \# z\odot = (y \# z)\odot$. Using this we can prove:

$$
\begin{aligned}
& Bag \circ (y\bigtimes) \# (z\bigtimes) \\
= \ & \{\text{definition } \_\bigtimes\} \\
& Bag \circ ([\![y\odot \triangledown \eta]\!]) \# ([\![z\odot \triangledown \eta]\!]) \\
= \ & \{\text{asso-sym rule: Theorem 6.6}\} \\
& Bag \circ ([\![(y\odot \# z\odot) \triangledown \eta]\!]) \\
= \ & \{\text{remark above}\} \\
& Bag \circ ([\![(y \# z)\odot \triangledown \eta]\!]) \\
= \ & \{y \# z \text{ point, definition } \_\bigtimes\} \\
& Bag \circ (y \# z)\bigtimes .
\end{aligned}
$$

From this it follows that

$$
\begin{aligned}
& Bag \circ \bigtimes y \circ \# = Bag \circ \# \circ \bigtimes y \times \bigtimes y \\
\equiv \ & \{\text{extensionality}\} \\
& Bag \circ \bigtimes y \circ \# \circ t \vartriangle s = Bag \circ \# \circ \bigtimes y \times \bigtimes y \circ t \vartriangle s \\
\equiv \ & \{\text{fusion } \vartriangle, \text{ definition } \_\#\_\} \\
& Bag \circ \bigtimes y \circ t \# s = Bag \circ (\bigtimes y \circ t) \# (\bigtimes y \circ s) \\
\equiv \ & \{\text{sectioning}\} \\
& Bag \circ (t \# s)\bigtimes \circ y = Bag \circ (t\bigtimes \circ y) \# (s\bigtimes \circ y) \\
\equiv \ & \{\text{split-imp fusion (38)}\} \\
& Bag \circ (t \# s)\bigtimes \circ y = Bag \circ (t\bigtimes) \# (s\bigtimes) \circ y \\
\Leftarrow \ & \{\text{calculation above}\} \\
& true . \qquad \Box
\end{aligned}
$$

The similarity between these computation rules and the normal computation rules suggests that cross-product is symmetric in a certain sense when we consider bags. If

we interchange the two arguments of $\bigtimes$, the arguments of the pairs of the result list get interchanged. Formally, we have the following theorem.

Theorem 6.15.

$$Bag \circ \bigtimes = Bag \circ \tilde{\bigtimes}_{\gamma}$$

where for a binary spec $\otimes$, $\tilde{\otimes}$ *is defined by* $\otimes \circ \gamma$.

Proof.

$$Bag \circ \bigtimes = Bag \circ \tilde{\bigtimes}_{\gamma},$$

$\equiv$ {definition $\tilde{\bigtimes}_{\gamma}$}

$$Bag \circ \bigtimes = Bag \circ *\gamma \circ \bigtimes \circ \gamma$$

$\equiv$ {extensionality}

$$Bag \circ \bigtimes \circ x \vartriangle I = Bag \circ *\gamma \circ \bigtimes \circ \gamma \circ x \vartriangle I$$

$\equiv$ {definition sectioning, property γ}

$$Bag \circ x\bigtimes = Bag \circ *\gamma \circ \bigtimes \circ I \vartriangle x$$

$\equiv$ {definition $x\bigtimes$, definition sectioning}

$$Bag \circ ([x\odot \vartriangle \eta]) = Bag \circ *\gamma \circ \bigtimes x$$

$\equiv$ {type-η fusion}

$$([(Bag \circ x\odot)\triangledown(Bag \circ \eta)]) = Bag \circ *\gamma \circ \bigtimes x$$

$\equiv$ {unique extension property}

$$Bag \circ *\gamma \circ \bigtimes x \circ \tau = Bag \circ x\odot$$

$$\wedge\ Bag \circ *\gamma \circ \bigtimes x \circ \Box = Bag \circ \Box$$

$$\wedge\ Bag \circ *\gamma \circ \bigtimes x \circ \# = Bag \circ \# \circ (Bag \circ *\gamma \circ \bigtimes x) \times (Bag \circ *\gamma \circ \bigtimes x).$$

The first two conjuncts follow from the cross-product computation rule of Theorem 6.14(a) and (b), and $\gamma \circ \gamma = I \times I$. For the last conjunct we have:

$$Bag \circ *\gamma \circ \bigtimes x \circ \#$$

$=$ {Theorem 5.2(c), cross-product computation: Theorem 6.14(c)}

$$Bag \circ *\gamma \circ \# \circ (\bigtimes x) \times (\bigtimes x).$$

$=$ {definition map, computation rule $\#$}

$$Bag \circ \# \circ *\gamma \times *\gamma \circ (\bigtimes x) \times (\bigtimes x)$$

$=$ {fusion $\times$}

$$Bag \circ \# \circ (*\gamma \circ \bigtimes x) \times (*\gamma \circ \bigtimes x)$$

$$
\begin{aligned}
&= \{\text{absorption rule}\}\\
&\quad Bag \circ \# \circ Bag \times Bag \circ (*\gamma \circ \bigtimes x) \times (*\gamma \circ \bigtimes x)\\
&= \{\text{fusion } \times\}\\
&\quad Bag \circ \# \circ (Bag \circ *\gamma \circ \bigtimes x) \times (Bag \circ *\gamma \circ \bigtimes x). \qquad \square
\end{aligned}
$$

A direct consequence of Theorem 6.15 in combination with Theorem 5.5(c) is the following corollary.

Corollary 6.15.

$$Bag \circ \bigtimes_{\otimes} = Bag \circ \widetilde{\bigtimes}_{\tilde{\otimes}}.$$

This theorem, which was suggested by Lambert Meertens, summarises everything we have proven for cross-product until now.

Using this corollary we can prove the Cartesian product rule:

Theorem 6.17 (Cartesian product). *If* $/\otimes$ *respects Bag then*

$$
\begin{aligned}
&/\otimes \circ *(/\otimes \circ x \bigtimes_{\otimes} \circ \tau) \circ y\\
&=\\
&/\otimes \circ *(/\otimes \circ \bigtimes_{\otimes} y \circ \tau) \circ x.
\end{aligned}
$$

Proof. First we prove

$$
\begin{aligned}
&/\otimes \circ x \bigtimes_{\otimes}\\
&= \{\text{definition } \bigtimes_{\otimes}\}\\
&\quad /\otimes \circ *\oplus \circ x \bigtimes\\
&= \{\text{definition } x\bigtimes\}\\
&\quad /\otimes \circ *\oplus \circ ([x \odot \vartriangle \eta])\\
&= \{\text{factorisation}\}\\
&\quad /\otimes \circ *\oplus \circ /\eta \circ *(x\odot)\\
&= \{/\eta \text{ leapfrog Theorem 3.9(b), } x\bigtimes \circ \tau = x.)\\
&\quad /\otimes \circ /\eta \circ **\oplus \circ *(x \bigtimes \circ \tau)\\
&= \{/\eta \text{ leapfrog Theorem 3.9(b), fusion map}\}\\
&\quad /\otimes \circ */\otimes \circ *(*\oplus \circ x \bigtimes \circ \tau)\\
&= \{\text{definition } \bigtimes_{\otimes}, \text{ fusion map}\}\\
&\quad /\otimes \circ *(/\otimes \circ x \bigtimes_{\otimes} \circ \tau).
\end{aligned}
$$

Using this and the assumption we derive the dual property:

$$
\begin{aligned}
& /\otimes \circ \bigtimes_{\otimes} y \\
= \ & \{/\otimes \text{ respects } \mathit{Bag}, \text{ Corollary 6.16}\} \\
& /\otimes \circ y \widetilde{\bigtimes}_{\tilde{\otimes}} \\
= \ & \{\text{property } \tilde{} \} \\
& /\otimes \circ y \bigtimes_{\tilde{\otimes}} \\
= \ & \{\text{calculation above}\} \\
& /\otimes \circ *(/\otimes \circ y \bigtimes_{\tilde{\otimes}} \circ \tau) \\
= \ & \{/\otimes \text{ respects } \mathit{Bag}, \text{ Corollary 6.16}\} \\
& /\otimes \circ *(/\otimes \circ y \widetilde{\bigtimes}_{\otimes} \circ \tau) \\
= \ & \{\text{property } \tilde{} \} \\
& /\otimes \circ *(/\otimes \circ \bigtimes_{\otimes} y \circ \tau).
\end{aligned}
$$

And combining the two gives us

$$
\begin{aligned}
& /\otimes \circ *(/\otimes \circ x \bigtimes_{\otimes} \circ \tau) \circ y \\
= \ & \{\text{first calculation}\} \\
& /\otimes \circ x \bigtimes_{\otimes} \circ y \\
= \ & \{\text{sectioning}\} \\
& /\otimes \circ \bigtimes_{\otimes} y \circ x \\
= \ & \{\text{second calculation}\} \\
& /\otimes \circ *(/\otimes \circ \bigtimes_{\otimes} y \circ \tau) \circ x. \qquad \square
\end{aligned}
$$

Within the quantifier calculus the Cartesian product rule looks much more familiar:

$$
\begin{aligned}
& \otimes(y \in A : q.y : \otimes(x \in B : p.x : x \oplus y)) \\
= & \\
& \otimes(x \in B : p.x : \otimes(y \in A : q.y : x \oplus y)).
\end{aligned}
$$

6.6. *Adding idempotence*

In this section we add idempotence, i.e. we consider sets. We know that $\mathit{Set} = \mathit{Set} \circ \mathit{Bag}$ so we may use the asso-sym rule (Theorem 6.6). Furthermore, we know that $\mathit{Set} \circ \#$ is idempotent, which gives us the following theorem.

Theorem 6.18 (Range disjunction). $\mathit{Set} \circ \triangleleft p \mathbin{\#\!\!\#} \triangleleft q = \mathit{Set} \circ \triangleleft(p \vee q)$.

Proof. First we prove:

$$
\begin{aligned}
& Set \circ (\tau \lhd p \rhd \varepsilon) \parallel (\tau \lhd q \rhd \varepsilon) \\
= \quad & \{\text{property conditionals: Theorem 3.13(b)}\} \\
& Set \circ ((\tau \parallel \tau) \lhd q \rhd (\tau \parallel \varepsilon)) \lhd p \rhd ((\varepsilon \parallel \tau) \lhd q \rhd (\varepsilon \parallel \varepsilon)) \\
= \quad & \{\text{property conditionals: Theorem 3.12(b), constructors: Definition 5.1(a), (b) and (e)}\} \\
& Set \circ (\tau \lhd q \rhd \tau) \lhd q \rhd (\tau \lhd q \rhd \varepsilon) \\
= \quad & \{\text{property conditionals: Theorem 3.12(a)}\} \\
& Set \circ \tau \lhd p \rhd (\tau \lhd q \rhd \varepsilon) \\
= \quad & \{\text{property conditionals: Theorem 3.12(g)}\} \\
& Set \circ \tau \lhd p \vee q \rhd \varepsilon .
\end{aligned}
$$

Then we have:

$$
\begin{aligned}
& Set \circ \lhd p \parallel \lhd q \\
= \quad & \{\text{definition filter}\} \\
& Set \circ ([\![\tau \lhd p \rhd \varepsilon \triangledown \eta]\!]) \parallel ([\![r \lhd q \rhd \varepsilon \triangledown \eta]\!]) \\
= \quad & \{\text{Theorem 6.6}\} \\
& Set \circ ([\![((\tau \lhd p \rhd \varepsilon) \parallel (\tau \lhd q \rhd \varepsilon)) \triangledown \eta]\!]) \\
= \quad & \{\text{type-}\eta\text{ fusion: Theorem 5.4, calculation above}\} \\
& Set \circ ([\![\tau \lhd p \vee q \rhd \varepsilon \triangledown \eta]\!]) \\
= \quad & \{\text{definition filter}\} \\
& Set \circ \lhd (p \vee q). \qquad \square
\end{aligned}
$$

Again we can combine this with a catamorphism to get the following theorem.

Theorem 6.19. *If* $([\![R \triangledown (S \triangledown \otimes)]\!])$ *respects Set then*

$$
\begin{aligned}
& ([\![R \triangledown (S \triangledown \otimes)]\!]) \circ \lhd (p \vee q) \\
= \quad & \\
& (([\![R \triangledown (S \triangledown \otimes)]\!]) \circ \lhd p) \otimes (([\![R \triangledown (S \triangledown \otimes)]\!]) \circ \lhd q).
\end{aligned}
$$

Proof.

$$
\begin{aligned}
&(\!|R \triangledown (S \triangledown \otimes)|\!) \circ \lhd (p \vee q) \\
= \;& \{(\!|R \triangledown (S \triangledown \otimes)|\!) \text{ respects } \mathit{Set}, \text{ range disjunction}\} \\
&(\!|R \triangledown (S \triangledown \otimes)|\!) \circ \lhd p \mathbin{\#} \lhd q \\
= \;& \{\text{definition } \_ \mathbin{\#} \_, \text{ computation rule } \#\} \\
&((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \lhd p) \otimes ((\!|R \triangledown (S \triangledown \otimes)|\!) \circ \lhd q). \qquad \square
\end{aligned}
$$

Theorem 6.19 stated in the quantifier notation yields

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

7. 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' 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. A surprising outcome is the low incidence of appeals to extensionality and/or functionality. Indeed, extensionality has only been used in the definition of the cross-product and recent work suggest that its use in this context can be eradicated.

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

$$\ll \circ R \vartriangle S = R. \tag{51}$$

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 (51) takes the form

$$\ll \circ R \vartriangle S = R \circ S{>}. \tag{52}$$

Our experience is that this has been a fortunate 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 "*List* ∘" or "*Bag* ∘" 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 that 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 every present.

Acknowledgements

We would like to thank Netty van Gasteren, Ed Voermans and Jaap van der Woude for their contributions to this work.

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

References

[1] C.J. Aarts, R.C. Backhouse, P.F. Hoogendijk, T.S. Voermans and J. van der Woude, A relational theory of datatypes (in preparation).

[2] R.C. Backhouse, *Program Construction and Verification* (Prentice-Hall International, Hemel Hempstead, England, 1986).

[3] R.C. Backhouse, An exploration of the Bird–Meertens formalism, Tech. Report CS8810, Department of Mathematics and Computing Science, University of Groningen, Netherlands (1988).

[4] R.C. Backhouse, P. de Bruin, P.F. Hoogendijk, G. Malcolm, T.S. Voermans and J. van der Woude, Polynomial relators, in: M. Nivat, C.S. Rattray, T. Rus and G. Scollo, eds., *Proceedings of the 2nd Conference on Algebraic Methodology and Software Technology, AMAST'91*, Workshops in Computing Series (Springer, Berlin, 1992) 303–362.

[5] R.C. Backhouse, P. de Bruin, G. Malcolm, T.S. Voermans and J. van der Woude, Relational catamorphisms, in: B. Möller, ed., *Proceedings of the IFIP TC2/WG2.1 Working Conference on Constructing Programs* (Elsevier Science Publishers B.V., Amsterdam, 1991) 287–318.

[6] R.C. Backhouse and H. Doornbos, Induction and recursion on datatypes, Department of Computing Science, Eindhoven University of Technology, Netherlands (1993).

[7] R.C. Backhouse and P.F. Hoogendijk, Elements of a relational theory of datatypes, in: B. Möller, H. Partsch and S. Schuman, eds., *Formal Program Development, IFIP TC2/WG2.1 State-of-the Art Report*, Lecture Notes in Computer Science **755** (Springer, Berlin, 1993) 7–42.

[8] J. Backus, Can programming be liberated from the von Neumann style? A functional style and its algebra of programs, *Comm. ACM* **21** (8) (1978) 613–641.

[9] R.S. Bird, The promotion and accumulation strategies in transformational programming, *ACM Trans. Programming Languages Syst.* **6** (4) (1984) 487–504.

[10] R.S. Bird, Transformational programming and the paragraph problem, *Sci. Comput. Programming* **6** (1986) 159–189.

[11] R.S. Bird, An introduction to the theory of lists, in: M. Broy, ed., *Logic of Programming and Calculi of Discrete Design*, NATO ASI Series F **36** (Springer, Berlin, 1987) 5–42.

[12] R.S. Bird, A calculus of functions for program derivation, Tech. Report, Programming Research Group, Oxford University (1988).

[13] R.S. Bird, Constructive functional programming, in: *Proceedings International Summer School on Constructive Methods in Computing Science*, Marktoberdorf, Germany (1988).
[14] R.S. Bird, Lectures on constructive functional programming, in: M. Broy, ed., *Constructive Methods in Computing Science*, NATO ASI Series F **55** (Springer, Berlin, 1989) 151–216.
[15] R.S. Bird, J. Gibbons and G. Jones, Formal derivation of a pattern matching algorithm, Tech. Report, Programming Research Group, Oxford University (1988).
[16] R.S. Bird and L.G.L.T. Meertens, Two exercises found in a book on algorithmics, in: L.G.L.T. Meertens, ed., *Program Specification and Transformations* (Elsevier Science Publishers B.V., North-Holland, Amsterdam, 1987) 451–457.
[17] M.M. Fokkinga, Law and order in algorithmics, Ph.D. Thesis, Universiteit Twente, Netherlands (1992).
[18] P.J. Freyd and A. Scedrov, *Categories, Allegories* (North-Holland, Amsterdam, 1990).
[19] C.A.R. Hoare, A couple of novelties in the propositional calculus, *Z. Math. Logik Grundl. Math.* **31** (2) (1985) 173–178.
[20] C.A.R. Hoare et al., Laws of programming, *Comm. ACM* **30** (8) (1987) 672–686; Corrigenda, *Comm. ACM* 30 (9) (1987) 770.
[21] K. Iverson, *A Programming Language* (Wiley, New York, 1962).
[22] G. Malcolm, Data structures and program transformation, *Sci. Comput. Programming* **14** (1990) 255–279.
[23] L.G.L.T. Meertens, Algorithmics—towards programming as a mathematical activity, in: *Proceedings CWI Symposium on Mathematics and Computer Science* (North-Holland, Amsterdam, 1986) 289–334.
[24] F.J. Rietman, A note on extensionality, in: J. van Leeuwen, ed., *Proceedings Computer Science in The Netherlands 91* (1981) 468–483.
[25] J. Riguet, Relations binaires, fermetures, correspondances de Galois, *Bull. Soc. Math. France* **76** (1948) 114–155.
[26] G. Schmidt and T. Ströhlein, *Relationene und Grafen* (Springer, Berlin, 1988).
[27] E. Voermans, Pers as types, inductive types and types with laws, in: *PHOENIX Seminar and Workshop on Declarative Programming, Sasbachwalden*, Workshops in Computing Series (Springer, Berlin, 1991).