# An Exploration of the Bird-Meertens Formalism

Roland Backhouse

Department of Computing Science
Groningen University
PO Box 800
9700 AV GRONINGEN
The Netherlands

July 15, 1988

**Abstract**

Two formalisms that have been used extensively in the last few years for the calculation of programs are the Eindhoven quantifier notation and the formalism developed by Bird and Meertens. Although the former has always been applied with ultimate goal the derivation of imperative programs and the latter with ultimate goal the derivation of functional programs there is a remarkable similarity in the formal games that are played. This paper explores the Bird-Meertens formalism by expressing and deriving within it the basic rules applicable in the Eindhoven quantifier notation.

> Calculation was an endless delight to Moorish scholars. They loved problems, they enjoyed finding ingenious methods to solve them, and sometimes they turned their methods into mechanical devices.
> (J. Bronowski, The Ascent of Man. Book Club Associates: London (1977).)

# 1 Introduction

Our ability to calculate — whether it be sums, products, differentials, integrals, or whatever — would be woefully inadequate, if not non-existent, were it not for the use of suitably chosen mathematical notation. When we perform a calculation, say $2167 \times 325$, the methods we use are purely syntactic, executed without regard to the meaning of the symbols “2”, “3” etc. Such formal manipulations of uninterpreted expressions are the key to economy and clarity, the two ingredients vital for effective reasoning.

Our ability to calculate computer programs is woefully inadequate and in many application areas non-existent largely because notations have yet to be developed that emphasise economy and manipulability; most programming notations, on the contrary, emphasise verbosity on the grounds of some ill-conceived notions of “readability” or, nowadays, “user-friendliness”.

The programming language APL designed by Iverson [17] was the first to emphasise economy of expression — the construction of APL “one-liners” was a favourite pastime some ten or more years ago — but so far as I am aware there was little emphasis on manipulability, that is on the recording and use of algebraic properties in order to simplify programs or make them more efficient. This aspect of notation was, however, strongly advocated by Backus in his Turing award lecture [5].

The present work grew out of a very personal endeavour. From my very first scientific publication onwards [2] the importance of algebraic manipulations in the construction of computer programs has been a dominant influence on my work. Relatively recently, whilst preparing my book entitled “Program Construction and Verification”, the major emphasis of my work was the development and application of a calculus of quantifications over finite domains. This work, which was inspired by a seminar given by David Gries somewhere around 1982, was unfortunately done in complete ignorance of similar work at Eindhoven University and documented in [15] and various privately-circulated documents written by E. W. Dijkstra and his colleagues. At the same time I was aware of the work of Richard Bird [8, 9] on an alternative calculus for program construction. The latter work had, for me, a clearly recognisable connection with my own work — in spite of the quite irrelevant fact that my implementation language was imperative whereas Bird's was recursively-defined functions — but I had no time to explore the connections in detail. Only recently did I make the time for such an exploration, spurred on by the observation that Bird together with Meertens [10, 6, 11, 20] seemed to have been making remarkable progress.

The goal that I set myself, the outcome of which is reported here, was self-evident. In my book [1] I had identified a number of properties of quantifications over finite domains that experience showed were central to the calculation of a number of programs. The question was, "could I express and derive all of these properties within the Bird-Meertens formalism?" Discovering the answer — "yes" with a few qualifications — turned out to be not so easy as I imagined, but certainly instructive.

Performing calculations is a private enjoyment that is difficult to share with others. Were I to include in this paper the details of the calculation of $2167 \times 325$ then I would be unlikely to persuade any reader to follow my working although I might be able to persuade some to repeat the calculations for themselves. Knuth [18] has complained about the lack of training in creative mathematics in his "anti-text" specifically intended to be "the exact opposite of Landau's *Grundlagen der Mathematik*". But I for one find reading his calculations just as boring (and just as interesting) as reading Landau's calculations. What is important, however, is whether the reader is enabled to reproduce some of the calculations and stimulated to invent and execute some of his own. This paper has therefore been written for my own enjoyment and education, but with a serious purpose. Its main significance is the inclusion of numerous calculations — not the hundred-odd results that are summarised in the appendix — but I fully expect that none of its readers will check every step in every calculation.

How then should the reader "read" and criticise the calculations? As mentioned before, the two guiding principles are clarity and economy. With regard to economy it is important to ascertain whether the calculations include unnecessary elements; it is also important to ascertain whether all the details are supplied or whether economy has been achieved artificially by brushing the details under the carpet. Economy has a great influence on clarity, both positive and negative. By the removal of all extraneous information the essential arguments are exposed, but the removal of too much information causes ambiguity and confusion.

Economy can of course be measured simply by counting symbols or lines but clarity is much more difficult to assess. The acid test that I would recommend to the reader at the detailed level of individual proof steps is to arbitrarily choose a sample of steps (taken from different proofs) and check each one independently of the remainder of the proof. How easy is it to parse the expressions involved? (An essential prerequisite but a monumentally difficult task in many mathematical documents!) Is it possible to quickly identify what the step entails? How difficult is it to verify the step from the given hints, and is the level of difficulty uniform throughout the text? Note that these tests should be performed entirely at the level of syntax. If you find yourself mentally trying to execute the two sides of an equation then you should give up at once. The calculations presented here would involve several man-years of effort had semantics been allowed to enter the picture.

The organisation of the calculations is just as important but is something for which it is less easy to prescribe guiding principles. The main principle — goal even — used in the preparation of this paper was the avoidance wherever possible of proofs by induction. This has been achieved by identifying powerful theorems, which are themselves proved by induction, and which are subsequently applied to calculate the remaining theorems. There are in fact just four theorems in the whole of this paper that have inductive proofs — the map distributivity law and the tree identity theorem in section 6, the promotability theorem in section 8 and the associativity and symmetry rule in section 12. To these must be added the inductive definitions of reduce and map in section 6. The fact that induction is needed so rarely was a very pleasant surprise to me; nevertheless I suspect that further improvements could be made.

A novelty of this paper is its discussion of the so-called "Boom type hierarchy" in converting properties of trees to properties of (finite) lists, bags and sets. The exploration of this hierarchy was a subsidiary rather than a primary goal of writing this paper although a clear account of the various theorems would be impossible without it. Somewhat against my own maxims I have therefore been guilty of making economies of presentation in reasoning about types that may bewilder the inexperienced reader.

I have refrained almost entirely from any comment on the serious purpose of this paper, namely its practical application to calculating computer programs. That there is indeed a practical relevance to the formal game played here is argued in the publications referenced above. In any case I suspect that anyone who remains unconvinced will not be reading this sentence anyway!

# 2 General Notational Remarks

The choice of appropriate notations is central to clarity and economy of program development; we take that as beyond dispute. Such choices should therefore be made explicitly and be justified. Conventionality or familiarity of a notation is, by the way, the weakest possible justification for a notation, at least in our view. Considerable thought has gone into the choice of notations in this paper but even then we are not sure that we have always made the right choice. Those places where we are doubtful will be made clear.

For the presentation of equational proofs we use the style introduced by W.H.J. Feijen in [15]. That is, we write

$$\begin{array}{lll} & P & \\ = & & \{ \text{ hint why } P = Q \ \} \\ & Q & \\ = & & \{ \text{ hint why } Q = R \ \} \\ & R & \end{array}$$

This format emphasises of course the transitivity of equality: all the expressions $P, Q$ and $R$ are equal, but in particular the first and last.

In a few cases equivalence ($\equiv$) or implication ($\Rrightarrow$) takes the place of equality. In such cases the connectives are used *conjunctively*; for example, $P \Rrightarrow Q \Rrightarrow R$ is understood as $(P \Rrightarrow Q)$ *and* $(Q \Rrightarrow R)$. (In other words the transitivity of equivalence and implication are emphasised by the notation.)

Our hints are composed as follows. First every formula, lemma, corollary and theorem has been numbered. Some of the more important ones have also been named; the others are grouped within subsections. Each hint then consists of the name given to a particular result or a name of a subsection together with the number of the result being applied. The intention is that readers who find it tedious to repeatedly turn back to previous sections may be able to recover the step for themselves from the supplied name alone. A name will be omitted only in the case that the formula referred to appears in the same subsection.

A summary has been provided in the appendix. This summary does not contain all the properties we prove but the main results of each section, including all those that are referenced in later sections. Having the appendix at hand when reading through the calculations is to be recommended.

For inductive proofs and proofs which involve case-analysis we use the Dijkstra-Feijen scope brackets ("|[" and "]|") to delimit the scope of an assumption. The symbolism

$$\begin{array}{ll} |[ & P \\ \triangleright & Q \\ ]| & \end{array}$$

should be read as "within the context of the assumption $P$ it follows that $Q$."

We endeavour to be highly systematic in our choice of identifiers. The identifiers $P, Q$ and $R$ just used denote arbitrary expressions. The symbols $A, B, C$ and $D$ will denote types. Other conventions will be introduced from time to time.

The type of functions with domain $A$ and range $B$ is denoted by $A \longrightarrow B$, and the cartesian product of $A$ and $B$ is denoted by $A \times B$.

We use prefix, infix and postfix notation for function application. For the moment we consider only prefix notation.

The application of function $f$ of type $A \longrightarrow B$ to $x$ of type $A$ is denoted with an infix dot, thus "$f.x$". The choice of such a small symbol (the smallest there is) means that we must give it the highest precedence: the eye sees, for example, $f + g.x$ as $f + (g.x)$, not as $(f + g).x$, and it would be foolish to try to persuade it otherwise.

We choose to accept conventional wisdom and postulate that function application associates to the left. I.e. $f.g.h$ denotes $(f.g).h$. Correspondingly the type former "$\longrightarrow$" associates to the right. Thus $A \longrightarrow B \longrightarrow C$ denotes $A \longrightarrow (B \longrightarrow C)$.

Function composition is denoted by the operator "$\circ$". On the grounds of experience with the manipulations that follow we choose to give composition the lowest possible precedence whenever it is used as an infix operator. (There is one exception to this rule, namely when it is used in combination with a conditional. This will be discussed in section 5. We also use it as a postfix operator in which case different rules apply. These rules are discussed shortly.) In order to compensate for the symbol's small size we include plenty of white space on either side of it. For example, we write $\oplus/ \ \circ \ f*$ and not $\oplus/\circ f*$. As is well known, function composition is associative so we can dispense with parentheses in an expression involving a succession of compositions.

To summarise, we have the following basic laws:

<u>Functions</u>

(1) $$h.x.y = (h.x).y$$

(2) $$(f \circ g).x = f.(g.x)$$

(3) $$f \circ (g \circ h) = (f \circ g) \circ h$$

Note that (3) can be proved from (2) by an extensionality argument. (Two functions are equal equivales they produce equal values when applied to an arbitrary argument.) It is at this level of detail that we terminate our proofs; thus, we state and subsequently exploit a number of formulae, like (3), that can be justified by unfolding a definition, computation rules and/or extensionality and leave the reader to verify their validity.

The omission of parentheses in an expression such as $f \circ g \circ h$ is worth reemphasising. An important characteristic of a good notation is that it should encourage the application of the most important algebraic properties by making their use (almost) invisible and, hence, (almost) automatic. In recognition of the importance of the associativity of function composition — it is applied in very many steps of our proofs — we have made its use (almost) invisible and nowhere do we refer to its use. Had we done otherwise our proofs would have almost doubled in length.

We use the symbols $\oplus$,$\otimes$ and $\odot$ to denote arbitrary binary operators. Application of such operators to their arguments is denoted using the usual infix notation, thus "$x \oplus y$". We use the symbols "$*$", "$/$", "$\triangleleft$" and "$+\!\!+$" to denote specific binary operators (called "map", "reduce", "filter" and "join", respectively) that are defined in later sections. The first three of these operators (map, reduce and filter) will always be used in their *curried* forms. Application of a curried operator to its first argument is denoted by juxtaposition, the argument preceding the operator. For example, map ("$*$") has two arguments the first of which is a function and the second is a list. Given function $f$ the application of map to $f$ is denoted by $f*$ and is a function having a list as argument. Application of $f*$ to the list $x$ is denoted by an infix dot, thus $f*.x$. The motivation for this choice is that we wish to regard $f*$ as an entity in its own right; in particular, we shall avoid as far as possible expressions in which it is applied to an argument. The same argument applies to reduce and filter.

In later sections we also find it necessary "curry" function composition. Indeed, we shall curry it in two ways. If $f$ is a function then we write $(f\circ)$ for the function such that $(f\circ).g = f \ \circ \ g$, and we write $(f\tilde{\circ})$ for the function such that $(f\tilde{\circ}).g = g \ \circ \ f$.

The above conventions are instances of a notational trick, called *sectioning* by Bird [7], whereby if $\odot$ has type $A{\times}B \longrightarrow C$ and $x$ has type $A$ then $(x\odot)$ denotes the function of type $B{\longrightarrow}C$ such that

$$(x\odot).y = x \odot y$$

for all $y$ of type $B$. We have deviated from Bird's conventions in two ways. First, by postulating that map, filter and reduce are always curried we avoid the irritation of always having to surround them by parentheses. (In practice, Bird also omits the parentheses under the assumption that the reader can glean the intended meaning from the context.) Second, we have chosen not to write $(\circ g)$ instead of $(g\tilde{\circ})$. Again this is to avoid excessive parenthesisation: for example the expression $(\circ f*)$ is confusing since it might be interpreted as either $((\circ f)*)$ or as $(\circ(f*))$. In fact, the rules we have given prohibit the first interpretation, but that may not be much consolation to an already bewildered reader!

(It should be remarked that not all ambiguity has been removed in this way. For example, "$\circ\circ\circ$" can be parsed as the infix expression of the form "$x \odot y$", where $x, y$ and $\odot$ are all "$\circ$", or as "$(\circ\circ)\circ$". Fortunately, such abstruse expressions never arise. In any case we make liberal use of white space around binary operators to ensure that the correct parsing of an expression is immediately obvious.)

The value of these notational tricks will be to observe unusual and unexpected distributivity properties. An example, of which the author was unaware before beginning this work, is that function composition distributes over itself:

$$(f\circ) \ \circ \ (g\circ) = (f \ \circ \ g)\circ$$

This is in fact just another way of stating that function composition is associative; but by restating associativity as a distributivity property we have the potential of discovering a whole host of further properties by exploiting theorems about distributivity.

The following rules summarise this discussion of infix operators and currying.

<u>Currying</u>

$$
\begin{array}{lrcl}
(4) & x\odot & = & \odot.x \\
(5) & (f\circ).g & = & f \circ g \\
(6) & (f\tilde{\circ}).g & = & g \circ f
\end{array}
$$

where $\odot$ is one of $*$, $\triangleleft$ or $/$.

Throughout the paper the symbol $\oplus$ will denote a binary operator of type $B{\times}B \longrightarrow B$. If a binary operator such as $\oplus$ has a unit element we denote this unit element by $1_\oplus$. Thus by definition we have the following.

<u>Unit</u>

$$
\begin{array}{lrcl}
(7) & y & = & 1_\oplus \oplus y \\
(8) & y & = & y \oplus 1_\oplus
\end{array}
$$

Note that the identity function is the unit of function composition and is denoted by $1_\circ$.

Our use of subscripts is confined to just this one case. We use subscripts rather than function application because there is no polymorphic function that given an operator $\oplus$ returns its unit element. When we discuss a particular operator we must therefore explicitly exhibit its unit.

A final remark. A minor inconvenience of our notational choices is that the function application symbol should separate all prefixed unary operators from their argument. For example, we have to write $\neg.p$ for the negation of predicate $p$ and $-.n$ for the negation of the number $n$. Time will tell whether or not we decide to introduce special cases.

# 3 Polymorphism and the Boom Hierarchy

A subsidiary aim of this paper is to experiment with the effective use of type information to economise on excess notation. This section therefore discusses notational issues to do with type.

## 3.1 Polymorphism

Many of the functions we deal with are polymorphic, function composition being one such. Letting $\mathcal{P}(A, B)$ denote the set of dependent polymorphic functions with domain $A$ and range $(\bigcup x \in A)B(x)$ we have the following introduction and elimination rules.

$$\frac{\begin{array}{ll} |[ & x \in A \\ \triangleright & f \in B(x) \\ ]| & \end{array}}{f \in \mathcal{P}(A,B)} \qquad\qquad \frac{\begin{array}{l} f \in \mathcal{P}(A,B) \\ a \in A \end{array}}{f \in B(a)} \qquad\qquad \text{polymorphism}$$

In these rules parenthesised occurrences of $x$ signify dependence on $x$ (not function application). The point to note is that in the introduction rule $f$ does *not* depend on $x$.

Typical examples of polymorphic functions are the identity function

$$1_\circ \ \in \ \mathcal{P}(\mathit{Type}, [\alpha]\alpha{\longrightarrow}\alpha)$$

and function composition

$$\circ \ \ \in \ \mathcal{P}(\mathit{Type}, [\alpha]\mathcal{P}(\mathit{Type}, [\beta]\mathcal{P}(\mathit{Type}, [\gamma]((\beta{\longrightarrow}\gamma) \times (\alpha{\longrightarrow}\beta)){\longrightarrow}\alpha{\longrightarrow}\gamma)))$$

Here *Type* denotes some universe of types and $[\alpha]$ denotes abstraction with respect to $\alpha$.

Polymorphism as discussed in Milner's seminal work [21] is restricted to type polymorphism, but the above rules are more general.

In order to avoid cumbersome notation we will always use $\alpha, \beta, \gamma$ for bound type variables and the brackets $[\![$ and $]\!]$ to indicate the binding. Thus the type of composition is expressed as follows.

$$\circ \ \ \in \ [\![((\beta{\longrightarrow}\gamma) \times (\alpha{\longrightarrow}\beta)){\longrightarrow}\alpha{\longrightarrow}\gamma]\!]$$

## 3.2 The Boom Hierarchy

If $A$ is a type then the type of all finite binary *trees* of elements of type $A$ is denoted by *Tree.A*. There are three ways a binary tree can be constructed. First, the empty tree, which we denote by $1_{+\!\!+}$, is a tree.

$$\frac{}{1_{+\!\!+} \in \mathit{Tree}.A} \quad 1_{+\!\!+}\text{-introduction}$$

Second, given element $a \in A$, one can form a tree consisting of just that element.

$$\frac{a \in A}{\tau.a \in \mathit{Tree}.A} \quad \tau\text{-introduction}$$

Finally, if $s$ and $t$ are trees then a tree with left subtree $s$ and right subtree $t$ can be formed with the join operation.

$$\frac{\begin{array}{l} s \in \mathit{Tree}.A \\ t \in \mathit{Tree}.A \end{array}}{s +\!\!+ t \in \mathit{Tree}.A} \quad +\!\!+\text{-introduction}$$

The reason for denoting the empty tree by $1_{+\!\!+}$ can now be explained. Joining an empty tree to a tree $t$ yields $t$.

$$\frac{t \in \mathit{Tree}.A}{t +\!\!+ 1_{+\!\!+} \ = \ 1_{+\!\!+} +\!\!+ t \ = \ t \ \in \ \mathit{Tree}.A} \quad \text{unit}$$

The judgement

$$t +\!\!+ 1_{+\!\!+} \ = \ 1_{+\!\!+} +\!\!+ t \ = \ t \ \in \ \mathit{Tree}.A$$

should be read as "within the type $\mathit{Tree}.A$ the elements $t +\!\!+ 1_{+\!\!+}$, $1_{+\!\!+} +\!\!+ t$ and $t$ are equal."

To reason about trees we use structural induction; to define a function over trees we also use structural induction — that is we define its value for the empty tree, for a tree of the form $\tau.a$ and for the join of two trees — but we have to be careful to ensure that the definition respects the above-mentioned equalities. In other words, for a function $f$ to be well-defined over trees we must check that

$$f.(1_{+\!\!+} +\!\!+ t) \ = \ f.(t +\!\!+ 1_{+\!\!+}) \ = \ f.t$$

for all trees $t$. (Typically, this is a task that we leave to the reader.)

A tree may also be regarded as a *list* if we postulate that the join operation is associative. Thus the following rules specify the construction of elements of type $\mathit{List}.A$.

$$\frac{}{1_{++} \in \mathit{List}.A} \quad 1_{++}\text{-introduction}$$

$$\frac{a \in A}{\tau.a \in \mathit{List}.A} \quad \tau\text{-introduction}$$

$$\frac{\begin{array}{l} s \in \mathit{List}.A \\ t \in \mathit{List}.A \end{array}}{s ++ t \in \mathit{List}.A} \quad ++\text{-introduction}$$

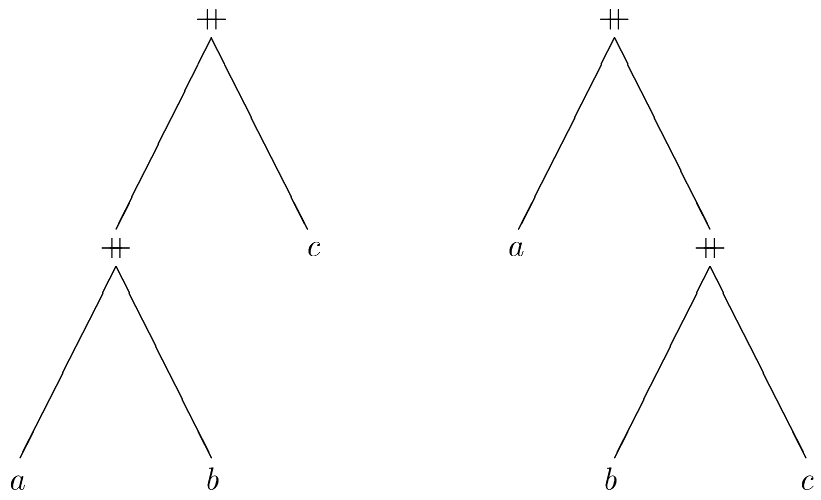
$$\frac{t \in \mathit{List}.A}{t ++ 1_{++} \;=\; 1_{++} ++ t \;=\; t \;\in\; \mathit{List}.A} \quad \text{unit}$$

$$\frac{\begin{array}{l} r \in \mathit{List}.A \\ s \in \mathit{List}.A \\ t \in \mathit{List}.A \end{array}}{r ++ (s ++ t) \;=\; (r ++ s) ++ t \;\in\; \mathit{List}.A} \quad \text{associativity of } ++$$

The notion that every tree is a list may be difficult to grasp and some explanation in terms of conventional notation may help. The tree denoted here by $(\tau.a ++ \tau.b) ++ \tau.c$ would often be depicted as in figure 1(a). It is of course different from the tree $\tau.a ++ (\tau.b ++ \tau.c)$ depicted in figure 1(b).

Now if we postulate that $++$ is associative the tree is "flattened": both trees depicted in figure 1 are equal *as lists* to $\tau.a ++ \tau.b ++ \tau.c$, the list more conventionally denoted by $[a, b, c]$. Our rules for list construction therefore express the notion that lists can be formed by joining (or "appending") lists together but the specific order in which the individual joins are performed is irrelevant.

A *bag* is a list in which we disregard the order of presentation of the elements. Thus the rules for constructing bags are obtained by replacing "*List*" by "*Bag*" in all the rules given so far (in exactly the same way as we obtained the first four rules for lists from those for trees) and adding the rule that join is symmetric:



(a) $(\tau.a +\!\!+ \tau.b) +\!\!+ \tau.c$ (b) $\tau.a +\!\!+ (\tau.b +\!\!+ \tau.c)$

Figure 1: Two trees that are equal as lists.

$$\frac{s \in \mathit{Bag}.A \qquad t \in \mathit{Bag}.A}{s +\!\!+ t \;=\; t +\!\!+ s \;\in\; \mathit{Bag}.A} \qquad \text{symmetry of } +\!\!+$$

Finally, a *set* is a bag in which repetitions of an element are disregarded, or, in other words, in which the join operation is idempotent. Thus the rules for constructing sets are obtained by replacing "*Bag*" by "*Set*" in the (six) rules for bags and by adding the rule:

$$\frac{t \in \mathit{Set}.A}{t +\!\!+ t \;=\; t \;\in\; \mathit{Set}.A} \qquad \text{idempotency of } +\!\!+$$

Note that our rules define *finite* structures, i.e., finite trees, finite lists, finite bags and finite sets.

Since any tree is a list it follows that any function of type $B \longrightarrow \mathit{Tree}.A$, say, also has type $B \longrightarrow \mathit{List}.A$. Similarly, any function of type $B \longrightarrow \mathit{List}.A$ also has type $B \longrightarrow \mathit{Bag}.A$, and any function of type $B \longrightarrow \mathit{Bag}.A$ also has type $B \longrightarrow Set.A$.

On the other hand a function of type $Tree.A \longrightarrow B$ does not necessarily have type $List.A \longrightarrow B$. For a function over trees to be well-defined over lists it must respect the associativity of join. I.e. we have the rule

$$\frac{\begin{array}{l} f \in Tree.A \longrightarrow B \\ [\![ \;\; r, s, t \in Tree.A \\ \triangleright \;\; f.((r +\!\!+ s) +\!\!+ t) \;=\; f.(r +\!\!+ (s +\!\!+ t)) \;\in\; B \\ ]\!] \end{array}}{f \in List.A \longrightarrow B}$$

Similarly a function over lists is also a function over bags only if it respects the symmetry of join, and a function over bags is a function over sets only if it respects the idempotency of join.

$$\frac{\begin{array}{l} f \in List.A \longrightarrow B \\ [\![ \;\; s, t \in List.A \\ \triangleright \;\; f.(s +\!\!+ t) \;=\; f.(t +\!\!+ s) \;\in\; B \\ ]\!] \end{array}}{f \in Bag.A \longrightarrow B}$$

$$\frac{\begin{array}{l} f \in Bag.A \longrightarrow B \\ [\![ \;\; t \in Bag.A \\ \triangleright \;\; f.(t +\!\!+ t) \;=\; f.t \;\in\; B \\ ]\!] \end{array}}{f \in Set.A \longrightarrow B}$$

The hierarchy of types that we have just described is attributed by Meertens [20] to H.J. Boom. It plays a central rôle in the development that follows. The formal properties stated here allude to a complete formalisation based on Martin-Löf's theory of types [19] originally developed for finite sets by Chisholm [12] and later extended by the author, Chisholm and Malcolm [3, 4].

Some readers may suffer disquiet over the fact that no notational distinction has been made between the constructors for trees, lists, bags and sets. (It is common practice to use [a] and $\{a\}$ instead of $\tau.a$ for lists and sets, respectively, and to use $\cup$ instead of $+\!\!+$ for set join. What the common notations for trees and bags are I do not know.) I have chosen not to do so in order not to have to rewrite properties that hold in more than one type. The penalty is that substitution of equals for equals requires much more care — two objects may be equal in one type but not in another! However that problem already occurs elsewhere (although it is normally brushed under the carpet). For example, suppose $f \in A \longrightarrow B$, $g \in B \longrightarrow C$ and $h \in B \longrightarrow D$. Suppose $h$ has an inverse denoted by $h^{-1}$. Then a step in a proof might be

$$\begin{array}{ll} & g \circ f \\ = & \qquad\qquad \{ \; h^{-1} \circ h = 1_\circ \in B \longrightarrow B \; \} \\ & g \circ h^{-1} \circ h \circ f \end{array}$$

Of course the equality $h^{-1} \circ h = 1_\circ$ is only valid within the type $B \longrightarrow B$. It would be quite wrong to reason as follows.

$$\begin{array}{ll} & g \circ f \\ = & \qquad\qquad \{ \; h \circ h^{-1} = 1_\circ \; \} \\ & g \circ h \circ h^{-1} \circ f \end{array}$$

The type information in the hint is therefore vital.

# 4 Constants and Lifting

Within the Bird-Meertens formalism our preference will go wherever possible to eliminating function application. The mechanism that is invariably used to do so is to "lift" functions to higher order. Generally if function $f$ has type $(B \times C \times \cdots \times Y) \longrightarrow Z$ it can be *lifted* to a function $h$ of type $[\![((\alpha \longrightarrow B) \times (\alpha \longrightarrow C) \times \cdots \times (\alpha \longrightarrow Y)) \longrightarrow (\alpha \longrightarrow Z)]\!]$ by defining $h.(b, c, \cdots, y).a = f.((b.a), (c.a), \cdots, (y.a))$ for all $a \in A$, $b \in A \longrightarrow B$, $c \in A \longrightarrow C$, $\ldots$, $y \in A \longrightarrow Y$. (Note that a function having $n$ arguments can be lifted in $2^n$ ways by first currying $k$ of its arguments ($k \leq n$) and then regarding it as a function of $n - k$ arguments.) We use this mechanism in three cases: functions with zero, one, or two arguments. In the second case it suffices to remark that the lifting operator is just composition. I.e. if $f$ has type $B \longrightarrow C$ then $(f\circ)$ has type $[\![(\alpha \longrightarrow B) \longrightarrow (\alpha \longrightarrow C)]\!]$ and has the desired definition. Let us therefore begin with functions of zero arguments, i.e. constants.

Each element $a$ of type $A$ can be "lifted" to a polymorphic constant function of type $[\![\alpha \longrightarrow A]\!]$ where

$$(9) \qquad a\bullet.x \;=\; a$$

The function $a\bullet$ has two useful properties. First, it is a left-zero of composition.

$$(10) \qquad a\bullet \;\circ\; f \;=\; a\bullet$$

whenever $f \in B \longrightarrow C$ for arbitrary type $C$. Second, in the presence of lifting, function application can always be replaced by function composition.

$$(11) \qquad f \;\circ\; a\bullet \;=\; (f.a)\bullet$$

whenever $f \in A \longrightarrow B$.

Last, but by no means least, equality of constants equivales the equality of the lifted constants.

$$(12) \qquad a = b \;\equiv\; a\bullet = b\bullet$$

For a binary operator $\oplus$ of type $B \times B \longrightarrow B$ we define the *lifted* operator $\oplus^{\wedge}$ of type $[\![((\alpha \longrightarrow B) \times (\alpha \longrightarrow B)) \longrightarrow (\alpha \longrightarrow B)]\!]$ by

$$(13) \qquad (f \oplus^{\wedge} g).x \;=\; f.x \oplus g.x$$

for all types $A$, all $f, g \in A \longrightarrow B$ and all $x \in A$.

A property of lifting that is easily established by extensionality is the distributivity law.

$$(14) \qquad (f \oplus^{\wedge} g) \;\circ\; h \;=\; (f \;\circ\; h) \oplus^{\wedge} (g \;\circ\; h)$$

for all types $A, B, C$, all $f, g \in A \longrightarrow B$ and all $h \in C \longrightarrow A$.

For the moment we observe only one property of the combination of lifting constants and lifting binary operators.

$$(15) \qquad 1_{\oplus^{\wedge}} \;=\; (1_{\oplus})\bullet$$

**Proof**

$$
\begin{array}{ll}
 & \mathbf{true} \\
\equiv & \qquad \{ \text{ unit (8) } \} \\
 & (1_\oplus)_\bullet = (1_\oplus)_\bullet \oplus^{\wedge} 1_{\oplus^\wedge} \\
\equiv & \qquad \{ \text{ extensionality, lifting (13) } \} \\
 & (1_\oplus)_\bullet.x = (1_\oplus)_\bullet.x \oplus 1_{\oplus^\wedge}.x \\
\equiv & \qquad \{ \text{ constants (9) } \} \\
 & (1_\oplus)_\bullet.x = 1_\oplus \oplus 1_{\oplus^\wedge}.x \\
\equiv & \qquad \{ \text{ unit (7) } \} \\
 & (1_\oplus)_\bullet.x = 1_{\oplus^\wedge}.x \\
\equiv & \qquad \{ \text{ extensionality } \} \\
 & (1_\oplus)_\bullet = 1_{\oplus^\wedge}
\end{array}
$$

(End of proof)

# 5 Conditionals

In this paper we have occasion to use only the simplest form of conditional statement, namely an **if-then-else** statement. If $b$ is a Boolean and $x$ and $y$ are both of the same type, $A$ say, then we define the conditional $x \langle\!\langle b \rangle\!\rangle\, y$ by

(16) $$x \langle\!\langle \mathbf{T} \rangle\!\rangle\, y \;=\; x$$

(17) $$x \langle\!\langle \mathbf{F} \rangle\!\rangle\, y \;=\; y$$

Here **T** and **F** are the canonical elements of the finite type *Bool*.

If $p$ is a predicate on the type $A$ (i.e. a function from $A$ to *Bool*) and $f$ and $g$ both have type $A{\longrightarrow}B$ for some $B$, then the conditional $f \;\langle p \rangle\; g$ of type $A{\longrightarrow}B$ is defined by

(18) $$(f \;\langle p \rangle\; g).x \;=\; f.x \;\langle\!\langle p.x \rangle\!\rangle\; g.x$$

for all $x$ in $A$. (Note that this equation expresses the fact that the ternary operator $\langle\,\rangle$ is obtained by lifting $\langle\!\langle\,\rangle\!\rangle$.)

The binary operators $\langle\!\langle b \rangle\!\rangle$ and $\langle p \rangle$ have lowest precedence of all (yet lower than composition). Thus, for example, an expression such as $h \circ f \;\langle p \rangle\; h \circ g$ in (31) below is parsed as $(h \circ f)\;\langle p \rangle\;(h \circ g)$.

We denote the boolean disjunction operator by **or** the boolean conjunction operator by **and** and the boolean negation operator by **not** (written as a prefix operator). (The reader who has just raised his eyebrows at the long-winded notation will be pleased to learn that the use of these symbols is confined almost exclusively to this section.) We introduce the following definitions.

$$\mathbf{true} = \mathbf{T}\bullet \tag{19}$$
$$\mathbf{false} = \mathbf{F}\bullet \tag{20}$$
$$\vee = \mathbf{or}\hat{\ } \tag{21}$$
$$\wedge = \mathbf{and}\hat{\ } \tag{22}$$
$$\neg = \mathbf{not}\circ \tag{23}$$

With these definitions the following are easily proved.

$$x \langle\!\langle b \rangle\!\rangle x = x \tag{24}$$
$$f \langle p \rangle f = f \tag{25}$$
$$f \langle \mathbf{true} \rangle g = f \tag{26}$$
$$f \langle \mathbf{false} \rangle g = g \tag{27}$$
$$f \langle \neg.p \rangle g = g \langle p \rangle f \tag{28}$$
$$f \langle p \wedge q \rangle g = (f \langle q \rangle g) \langle p \rangle g \tag{29}$$
$$f \langle p \vee q \rangle g = f \langle p \rangle (f \langle q \rangle g) \tag{30}$$
$$h \circ (f \langle p \rangle g) = h \circ f \langle p \rangle h \circ g \tag{31}$$
$$(f \langle p \rangle g) \circ h = f \circ h \langle p \circ h \rangle g \circ h \tag{32}$$

Note that equation (25) assumes that all predicates are total.

At this point it is necessary to express some misgivings about the notation used for conditionals. The notation chosen emphasises that $\langle p \rangle$ is a binary operator, but does not allow the same amount of economy as notation used elsewhere in the paper. This is apparent, for example, in equation (32) where the variables play no real rôle. (The property that one type of conditional is the lift of the other is another example which cannot be expressed without the introduction of unnecessary variables.) Relatively few of our calculations, however, involve conditionals so we shall leave the problem aside.

# 6 Map, Reduce and Filter

The next step in this development is to define functions *map* (denoted by infix "$*$"), *reduce* (denoted by infix "/") and *filter* (denoted by infix "$\triangleleft$"). The choice of symbols is adopted from Bird [7]; the functions map and reduce appear in many accounts of functional programming and I do not know to whom they, and filter, should be credited. These three operators are of fundamental importance; in the next section we use them to reexpress the quantifier notation introduced by Dijkstra [14], and subsequently used extensively by Rem in his regular column in the journal Science of Computer Programming, by Gries [16], by Dijkstra and Feijen [15], by Backhouse [1] and by many others.

### Map

The function *map* is defined as follows.

$$
\begin{array}{ll}
* \in [\![(\alpha \longrightarrow \beta) \longrightarrow \mathit{Tree}.\alpha \longrightarrow \mathit{Tree}.\beta]\!] & \\
f*.1_{+\!\!+} \;=\; 1_{+\!\!+} & \text{(empty rule)} \\
f*.(\tau.a) \;=\; \tau.(f.a) & \text{(one-pt. rule)} \\
f*.(s +\!\!+ t) \;=\; (f*.s) +\!\!+ (f*.t) & \text{(join rule)}
\end{array}
$$

(An operational understanding of map is that it applies its first argument, $f$, to each of the leaves of its second argument whilst retaining the same tree structure.)

For the moment the type information about map is minimal. Later we make stronger statements.

Map has the very important property that it distributes over function composition:

(33) $$(f \circ g)* \;=\; f* \circ g*$$

whenever $g \in A \longrightarrow B$ and $f \in B \longrightarrow C$ for some types $A, B$ and $C$.

We refer to (33) by the name *map distributivity.* Its proof is an easy induction proof and is left to the reader. So far as I am aware the first person to observe the law was Backus [5].

### Reduce

The function *reduce* is defined as follows:

$$/ \in [\![(\alpha\times\alpha \longrightarrow \alpha)\longrightarrow Tree.\alpha\longrightarrow\alpha]\!]$$
$$\oplus/.1_{++} \;=\; 1_{\oplus} \qquad \text{(empty rule)}$$
$$\oplus/.(\tau.a) \;=\; a \qquad \text{(one-pt. rule)}$$
$$\oplus/.(s +\!\!+\, t) \;=\; (\oplus/.s) \oplus \oplus/.t) \qquad \text{(join rule)}$$

Functions of the form $\oplus/ \circ f*$ are the dominant concern of this paper. An important property of such functions is that they maintain the algebraic properties of the operator $\oplus$:

**Theorem 34 (Type Hierarchy Theorem)** *Suppose* $f \in A\longrightarrow B$ *and* $\oplus \in B\times B \longrightarrow B$. *Then* $\oplus/ \circ f* \in Tree.A\longrightarrow B$. *Moreover,* $\oplus/ \circ f* \in List.A\longrightarrow B$ *whenever* $\oplus$ *is associative;* $\oplus/ \circ f* \in Bag.A\longrightarrow B$ *whenever* $\oplus$ *is, in addition, symmetric; finally,* $\oplus/ \circ f* \in Set.A\longrightarrow B$ *whenever* $\oplus$*, in addition to being associative and symmetric, is idempotent.*

**Proof**

We note the following.

$$\begin{array}{ll}
 & (\oplus/ \circ f*).(x +\!\!+\, y) \\
= & \qquad \{ \text{ functions (2), join rule for } * \} \\
 & \oplus/.((f*.x) +\!\!+\, (f*.y)) \\
= & \qquad \{ \text{ join rule for } / \} \\
 & (\oplus/.(f*.x)) \oplus (\oplus/.(f*.y)) \\
= & \qquad \{ \text{ functions (2) } \} \\
 & (\oplus/ \circ f*).x \oplus (\oplus/ \circ f*).y
\end{array}$$

It follows that $\oplus/ \circ f*$ maintains the associativity of $+\!\!+$ so long as $\oplus$ is associative, and similarly for symmetry and idempotence.
(End of proof)

**Corollary 35** *For* $h \in A\longrightarrow Tree.B$

$$\begin{array}{lcl}
+\!\!+/ \circ h* & \in & Tree.A\longrightarrow Tree.B \\
+\!\!+/ \circ h* & \in & List.A\longrightarrow List.B \\
+\!\!+/ \circ h* & \in & Bag.A\longrightarrow Bag.B \\
+\!\!+/ \circ h* & \in & Set.A\longrightarrow Set.B
\end{array}$$

**Proof**

The corollary follows immediately from the type hierarchy theorem and the observations that join is associative in $List.B$, it is associative and symmetric in $Bag.B$, and it is associative, symmetric and idempotent in $Set.B$ (End of proof)

**Corollary 36**

**(a)** *If* $\oplus \in B{\times}B \longrightarrow B$ *is associative then* $\oplus/ \in List.B{\longrightarrow}B$
**(b)** *If* $\oplus \in B{\times}B \longrightarrow B$ *is associative and symmetric then* $\oplus/ \in Bag.B{\longrightarrow}B$
**(c)** *If* $\oplus \in B{\times}B \longrightarrow B$ *is associative, symmetric and idempotent then* $\oplus/ \in Set.B{\longrightarrow}B$.

**Proof**

A proof can be given from first principles without using the type hierarchy theorem. Alternatively, anticipating equation (76) proved in section 9, we have

$$\oplus/ \;=\; \oplus/ \,\circ\, 1_\circ * \;\in\; Tree.B{\longrightarrow}B$$

Thus the corollary is an instance of the type hierarchy theorem.
(End of proof)

Often the most obvious properties are overlooked. The following theorem is "obvious", so much so that I have never seen it stated in its barest form. Yet it is one that we will apply frequently.

**Theorem 37 (Tree Identity Theorem)**

$$+\!\!\!+/ \circ \tau* \;=\; 1_\circ \;\in\; [\![Tree.\alpha{\longrightarrow}Tree.\alpha]\!]$$

**Proof**

By structural induction and extensionality.

**(a)**

$$\begin{array}{ll}
 & (+\!\!\!+/ \circ \tau*).1_{+\!\!\!+} \\
= & \qquad\{\text{ functions (2) }\} \\
 & +\!\!\!+/.(\tau*.1_{+\!\!\!+}) \\
= & \qquad\{\text{ empty rule for map }\} \\
 & +\!\!\!+/.1_{+\!\!\!+} \\
= & \qquad\{\text{ empty rule for reduce }\} \\
 & 1_{+\!\!\!+}
\end{array}$$

**(b)** $(+\!\!+/ \circ \tau*).(\tau.a)$

$=$ { functions (2), one-pt. rule for map }

$+\!\!+/.(\tau.(\tau.a))$

$=$ { one-pt. rule for reduce }

$\tau.a$

**(c)**

$|[$ $s, t \in \mathit{Tree}.A;\ \ (+\!\!+/ \circ \tau*).s = s;\ \ (+\!\!+/ \circ \tau*).t = t$

$\triangleright$ $(+\!\!+/ \circ \tau*).(s +\!\!+ t)$

$=$ { functions (2), join rule for map }

$+\!\!+/.(\tau*.s +\!\!+ \tau*.t)$

$=$ { join rule for reduce }

$+\!\!+/.(\tau*.s) +\!\!+ +\!\!+/.(\tau*.t)$

$=$ { functions (2), inductive hypothesis }

$s +\!\!+ t$

$]|$

It follows that $+\!\!+/ \circ \tau* = 1_\circ \in \mathit{Tree}.A \longrightarrow \mathit{Tree}.A$, whence we conclude the theorem by abstracting over $A$.
(End of proof)

**Corollary 38**

$$\begin{aligned}
+\!\!+/ \circ \tau* &= 1_\circ \in [\![\mathit{List}.\alpha \longrightarrow \mathit{List}.\alpha]\!] \\
+\!\!+/ \circ \tau* &= 1_\circ \in [\![\mathit{Bag}.\alpha \longrightarrow \mathit{Bag}.\alpha]\!] \\
+\!\!+/ \circ \tau* &= 1_\circ \in [\![\mathit{Set}.\alpha \longrightarrow \mathit{Set}.\alpha]\!]
\end{aligned}$$

**Proof**

Immediate from corollary 35 and the proof of theorem 37.
(End of proof)

The tree identity theorem was expressed by Bird [7] in the following slightly more general form.

**Corollary 39**

$$f* = +\!\!+/ \circ (\tau \circ f)*$$

**Proof**

$$
\begin{array}{ll}
 & f* \\
= & \qquad \{ \text{ unit (7) } \} \\
 & 1_\circ \circ f* \\
= & \qquad \{ \text{ tree identity (37) } \} \\
 & +\!\!+/ \circ \tau* \circ f* \\
= & \qquad \{ \text{ map distributivity (33) } \} \\
 & +\!\!+/ \circ (\tau \circ f)*
\end{array}
$$

(End of proof)

**Corollary 40**

$$
\begin{array}{rcl}
* & \in & [\![(\alpha \longrightarrow \beta) \longrightarrow List.\alpha \longrightarrow List.\beta]\!] \\
* & \in & [\![(\alpha \longrightarrow \beta) \longrightarrow Bag.\alpha \longrightarrow Bag.\beta]\!] \\
* & \in & [\![(\alpha \longrightarrow \beta) \longrightarrow Set.\alpha \longrightarrow Set.\beta]\!]
\end{array}
$$

**Proof**

Immediate from corollaries 35 and 39.
(End of proof)

**Filter**

The function *filter* is defined as follows.

$$\lhd \in [\![(\alpha \longrightarrow Bool) \longrightarrow Tree.\alpha \longrightarrow Tree.\alpha]\!]$$

(41) $$p\lhd \;=\; +\!\!+/ \circ (\tau \,\langle p \rangle\, 1_{+\!\!+^\wedge})*$$

An operational understanding of filter can be gained by comparing its definition with the tree-identity theorem. Its effect is to filter out all leaves of the tree not satisfying $p$ by replacing them by $1_{+\!\!+}$.

The tree identity theorem has the following corollary.

(42) $$\mathbf{true}\lhd \;=\; 1_\circ$$

**Proof**

$$
\begin{array}{ll}
 & \mathbf{true}\triangleleft \\
= & \qquad \{ \text{ filter } (41) \ \} \\
 & +\!\!\!+/ \circ (\tau \ \langle \mathbf{true} \rangle \ 1_{+\!\!\!+^{\wedge}})* \\
= & \qquad \{ \text{ conditionals } (26) \ \} \\
 & +\!\!\!+/ \circ \tau* \\
= & \qquad \{ \text{ tree identity } (37) \ \} \\
 & 1_{\circ}
\end{array}
$$

(End of proof)

We postpone the consideration of **false**$\triangleleft$ until later.

The following theorem follows from the definition (41) of filter and corollary 35.

**Theorem 43 (Functionality of filter)**

$$
\begin{array}{lcl}
\triangleleft & \in & [\![(\alpha \longrightarrow Bool) \longrightarrow Tree.\alpha \longrightarrow Tree.\alpha]\!] \\
\triangleleft & \in & [\![(\alpha \longrightarrow Bool) \longrightarrow List.\alpha \longrightarrow List.\alpha]\!] \\
\triangleleft & \in & [\![(\alpha \longrightarrow Bool) \longrightarrow Bag.\alpha \longrightarrow Bag.\alpha]\!] \\
\triangleleft & \in & [\![(\alpha \longrightarrow Bool) \longrightarrow Set.\alpha \longrightarrow Set.\alpha]\!]
\end{array}
$$

Now that our repertoire of functions and properties is beginning to grow we will put into practice our stated preference for avoiding function application (and thus eliminating some variables). To facilitate this we conclude this section with a summary of the basic properties of map, reduce and filter expressed in functional form.

**Empty Rules**

(44) $$f* \circ 1_{+\!\!\!+^{\wedge}} = 1_{+\!\!\!+^{\wedge}}$$

(45) $$\oplus/ \circ 1_{+\!\!\!+^{\wedge}} = 1_{\oplus^{\wedge}}$$

(46) $$p\triangleleft \circ 1_{+\!\!\!+^{\wedge}} = 1_{+\!\!\!+^{\wedge}}$$

**One-point Rules**

(47) $$f* \circ \tau = \tau \circ f$$

(48) $$\oplus/ \circ \tau = 1_{\circ}$$

(49) $$p\triangleleft \circ \tau = \tau \ \langle p \rangle \ 1_{+\!\!\!+^{\wedge}}$$

We leave the join rules in their existing form. (After the section on promotability they will be used less and less often.)

**Join Rules**

$$(50) \qquad f{*}.(s +\!\!+\, t) \;=\; (f{*}.s) +\!\!+ (f{*}.t)$$

$$(51) \qquad \oplus/.(s +\!\!+\, t) \;=\; (\oplus/.s) \oplus (\oplus/.t)$$

$$(52) \qquad p{\triangleleft}.(s +\!\!+\, t) \;=\; (p{\triangleleft}.s) +\!\!+ (p{\triangleleft}.t)$$

Of course most of these properties require proof, particularly the properties (46), (49) and (52) of filter which aren't simply rewrites of the definition. We prove (49) as an example.

**Proof of (49)**

$$\begin{array}{ll}
 & p{\triangleleft} \circ \tau \\
= & \qquad \{ \text{ filter (41) } \} \\
 & +\!\!+/ \circ (\tau\langle p\rangle 1_{+\!\!+^\wedge}){*} \circ \tau \\
= & \qquad \{ \text{ one-point rule (47) } \} \\
 & +\!\!+/ \circ \tau \circ (\tau\langle p\rangle 1_{+\!\!+^\wedge}) \\
= & \qquad \{ \text{ one-point rule (48) } \} \\
 & 1_\circ \circ (\tau\langle p\rangle 1_{+\!\!+^\wedge}) \\
= & \qquad \{ \text{ unit (7) } \} \\
 & \tau\langle p\rangle 1_{+\!\!+^\wedge}
\end{array}$$

(End of proof)

Note the extreme compactness of this proof! Note also how often associativity of function composition has been used! This proof gives a foretaste of the extraordinary advantages of avoiding the use of variables.

For later use we mention one more consequence of the definition of reduce obtained by combining the join and one-point rules.

$$(53) \qquad x \oplus y \;=\; \oplus/.(\tau.x +\!\!+ \tau.y)$$

# 7 The Eindhoven Quantifier Notation

We are now in a position to compare, at a descriptive level, the Eindhoven quantifier notation with the Bird-Meertens formalism. Specifically, if $\oplus$ is an arbitrary binary operator of type $B{\times}B \longrightarrow B$, $f$ is a function of type $A{\longrightarrow}B$ and $p$ is a predicate of type $A{\longrightarrow}Bool$ then, by definition,

$$\oplus(i : p.i : f.i) \;=\; \oplus/ \;\circ\; f* \;\circ\; p\triangleleft$$

It is vital to note that the expression $\oplus(i : p.i : f.i)$ 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 $\oplus(i \in A : p.i : f.i)$; if we now suppose that $\mathcal{A}$ is an enumeration of the elements of type $A$ and $\oplus$ is both associative and symmetric then we can make the definition

$$\oplus(i \in A : p.i : f.i) \;=\; (\oplus/ \;\circ\; f* \;\circ\; p\triangleleft).\mathcal{A}$$

(Note that the latter definition offers some justification for the prerequisite, within the Eindhoven school, that $\oplus$ be associative and symmetric for a quantification over $\oplus$ to be meaningful. It came as a surprise to the author during the writing of this paper to discover that that assumption is not necessary and a substantial body of properties can be expressed about trees.)

Our plan in the forthcoming sections is to formally express and prove as many as possible of the quantifier rules stated in [1] within the Bird-Meertens formalism. In many cases a literal translation of the quantifier rule is not particularly valuable; more often than not the quantifier rule has a more compact representation within the Bird-Meertens formalism from which the literal restatement is a straightforward corollary. There are some quantifier rules however for which we are unable to find an equivalent. One such is the one-point rule:

$$\oplus(i \in A : i = k : f.i) \;=\; f.k$$

which relies on implicit information about the type of the expression $k$. Another is the rule of range translation: for associative and symmetric $\oplus$ and bijection $t \in C{\longrightarrow}A$

$$\oplus(i \in A : p.i : f.i) \;=\; \oplus(j \in C : p.(t.j) : f.(t.j))$$

and for associative, symmetric and idempotent $\oplus$ and surjection $t \in C{\longrightarrow}A$

$$\oplus(i \in A : p.i : f.i) \;=\; \oplus(j \in C : p.(t.j) : f.(t.j))$$

On the other hand there are many properties provable within the Bird-Meertens formalism that are essential to its practical application but cannot even be expressed within the quantifier notation. This paper, therefore, explores the overlap; an exploration of their combination has yet to be undertaken.

Anticipating some later results we can write an expression of the form $\oplus/ \circ f* \circ p\triangleleft$ into one of the form $\oplus/ \circ g*$ as follows.

$$\oplus/ \circ f* \circ p\triangleleft \;=\; \oplus/ \circ (f\langle p\rangle 1_{\oplus^\wedge})*$$

Thus in theory the introduction of filter is superfluous; in practice, however, it adds considerably to the clarity of the expressions we write. The pertinence of the above observation is that we may apply the type hierarchy theorem.

**Theorem 54 (Type of quantifiers)** *For $\oplus$ of type $B{\times}B \longrightarrow B$, $f$ of type $A{\longrightarrow}B$ and $p$ of type $A{\longrightarrow}Bool$ the expression $\oplus(i : p.i : f.i)$ (or equally $\oplus/ \circ f* \circ p\triangleleft$) has type $Tree.A{\longrightarrow}B$. In addition, it has type*

**(a)** *$List.A{\longrightarrow}B$ if $\oplus$ is associative,*

**(b)** *$Bag.A{\longrightarrow}B$ if $\oplus$ is associative and symmetric,*

**(c)** *$Set.A{\longrightarrow}B$ if $\oplus$ is associative, symmetric and idempotent.*

# 8 Promotability

The list of definitions and new notations is almost at an end! We begin our exploration of the Bird-Meertens formalism, perhaps surprisingly, with the last of the rules in chapter 2 of [1], those concerned with so-called "generalised distributivity". Here we refer to the properties as **promotability** properties using the term coined by Bird [7]. The notion of promotability is ubiquitous in our proofs; it is truly enchanting that so many other properties can be derived from it.

The subsequent development follows the Boom type-hierarchy: first we consider trees, then lists, then bags and finally sets. Promotability is a property of trees (and thus of lists, bags and sets). From now on we work within the domain of trees unless specific mention otherwise is made. (In fact only a small number of properties we discuss are peculiar to bags and sets; these are discussed in the last two sections. We discuss no properties particular to lists.)

In this section we suppose that $\oplus \in B{\times}B \longrightarrow B$, $\otimes \in C{\times}C \longrightarrow C$ and $f \in B{\longrightarrow}C$. We begin by defining distributivity and promotability.

**Definition 55** *The function $f$ is said to be $\oplus{\rightarrow}\otimes$-distributive if and only if*

$$f.(x \oplus y) \;=\; (f.x) \otimes (f.y)$$

*for all $x, y \in B$.*

**Definition 56** *The function $f$ is said to be $\oplus{\rightarrow}\otimes$-promotable if and only if it is $\oplus{\rightarrow}\otimes$-distributive and also $f.1_\oplus = 1_\otimes$.*

Note that, since equality is typed, properties such as distributivity and promotability are also typed. In particular, a function $h$ may have the property of being $\oplus{\rightarrow}+\!\!\!+$-distributive (for some $\oplus$) when $+\!\!\!+$ is symmetric but not when $+\!\!\!+$ is asymmetric. The most obvious example is the function $rev$ that reverses the left and right subtrees of a tree; $rev.(x +\!\!\!+ y)$ is by definition $(rev.y) +\!\!\!+ (rev.x)$ and so within a type $Bag.A$, but not within $Tree.A$ or $List.A$, we have the equality $rev.(x +\!\!\!+ y) = (rev.x) +\!\!\!+ (rev.y)$. Similarly, a function may possess a distributivity property when $+\!\!\!+$ is idempotent but not otherwise, the simplest examples being functions of the form $(\tau.a)\bullet$ for arbitrary $a$ of type $A$. Within the type $[\![\alpha{\longrightarrow}Set.A]\!]$ all such functions are $+\!\!\!+{\rightarrow}+\!\!\!+$-distributive but within, for example, $[\![\alpha{\longrightarrow}Bag.A]\!]$ they are not.

In [1] promotability was called distributivity thus making the terminology out of line with common mathematical usage; the invention of a new word obviates this difficulty. The significance of promotability is captured by the following theorem.

**Theorem 57 (Promotability theorem)**
*The function $f$ is $\oplus{\rightarrow}\otimes$-promotable iff*

$$f \,\circ\, \oplus/ \;=\; \otimes/ \,\circ\, f*$$

**Proof**
First observe that

$$
\begin{array}{lll}
(\dagger) & & (f \circ \oplus/ \circ 1_{+\!\!+^\wedge} \;=\; \otimes/ \circ f* \circ 1_{+\!\!+^\wedge}) \\
 & \equiv & \qquad\{ \text{ empty rules (44), (45) } \} \\
 & & (f \circ 1_{\oplus^\wedge} \;=\; 1_{\otimes^\wedge})
\end{array}
$$

Now,

$$
\begin{array}{lll}
|[ & f \circ \oplus/ \;=\; \otimes/ \circ f* & \\
\triangleright & \quad \{(\dagger), \text{constants and lifting (15), (11), (12)}\} & \\
 & f \circ 1_{\oplus\hat{}} \;=\; 1_{\otimes\hat{}} & \\
; & \quad f.(x \oplus y) \;=\; f.x \otimes f.y & \\
 & \equiv \qquad \{ \text{functions (2), join rule (53)} \} & \\
 & \quad (f \circ \oplus/).(\tau.x \mathbin{+\!\!\!+} \tau.y) \;=\; \otimes/.(\tau.(f.x) \mathbin{+\!\!\!+} \tau.(f.y)) & \\
 & \equiv \qquad \{ \text{functions (2), one-pt. rule (47)} \} & \\
 & \quad (f \circ \oplus/).(\tau.x \mathbin{+\!\!\!+} \tau.y) \;=\; \otimes/.((f* \circ \tau).x \mathbin{+\!\!\!+} (f* \circ \tau).y) & \\
 & \equiv \qquad \{ \text{join rule (50), functions (2)} \} & \\
 & \quad (f \circ \oplus/).(\tau.x \mathbin{+\!\!\!+} \tau.y) \;=\; (\otimes/ \circ f*).(\tau.x \mathbin{+\!\!\!+} \tau.y) & \\
 & \equiv \qquad \{ \text{assumption} \} & \\
 & \quad \mathbf{true} & \\
]| & &
\end{array}
$$

I.e. if $f \circ \oplus/ \;=\; \otimes/ \circ f*$ then $f$ is $\oplus{\rightarrow}\otimes$-promotable .
Also,

$$
\begin{array}{lll}
|[ & f \text{ is } \oplus{\rightarrow}\otimes\text{-promotable} & \\
\triangleright & \quad \{(\dagger)\} & \\
 & f \circ \oplus/ \circ 1_{+\!\!\!+\hat{}} \;=\; \otimes/ \circ f* \circ 1_{+\!\!\!+\hat{}} & \\
; & \quad f \circ \oplus/ \circ \tau \;=\; \otimes/ \circ f* \circ \tau & \\
 & \equiv \qquad \{ \text{one-pt. rules (48), (47)} \} & \\
 & \quad f \circ 1_{\circ} \;=\; 1_{\circ} \circ f & \\
 & \equiv \qquad \{ \text{unit (7) and (8), reflexivity} \} & \\
 & \quad \mathbf{true} & \\
; & |[ \quad (f \circ \oplus/).x \;=\; (\otimes/ \circ f*).x & \\
 & ; \quad (f \circ \oplus/).y \;=\; (\otimes/ \circ f*).y & \\
 & \triangleright \quad (f \circ \oplus/).(x \mathbin{+\!\!\!+} y) \;=\; (\otimes/ \circ f*).(x \mathbin{+\!\!\!+} y) & \\
 & \quad \equiv \qquad \{ \text{join rules (50), (51), functions (2)} \} & \\
 & \quad f.((\oplus/.x) \oplus (\oplus/.y)) \;=\; (\otimes/ \circ f*).x \otimes (\otimes/ \circ f*).y & \\
 & \quad \equiv \qquad \{ f \text{ is } \oplus{\rightarrow}\otimes\text{-promotable, functions (2)} \} & \\
 & \quad (f \circ \oplus/).x \otimes (f \circ \oplus/).y \;=\; (\otimes/ \circ f*).x \otimes (\otimes/ \circ f*).y & \\
 & \quad \equiv \qquad \{ \text{inductive hypotheses, reflexivity} \} & \\
 & \quad \mathbf{true} & \\
 & ]| &
\end{array}
$$

]|

It thus follows by the Tree elimination rule that if $f$ is $\oplus{\rightarrow}\otimes$-promotable then $f \circ \oplus/ = \otimes/ \circ f*$.
(End of proof)

Some familiar examples of promotability are given in the following corollaries of the promotability theorem.

(58) $$\mathbf{not} \circ \mathbf{and}/ = \mathbf{or}/ \circ \mathbf{not}*$$

(59) $$\mathbf{not} \circ \mathbf{or}/ = \mathbf{and}/ \circ \mathbf{not}*$$

(60) $$- \circ \mathit{min}/ = \mathit{max}/ \circ -*$$

(61) $$- \circ \mathit{max}/ = \mathit{min}/ \circ -*$$

(62) $$k+ \circ \mathit{min}/ = \mathit{min}/ \circ k+*$$

(63) $$k+ \circ \mathit{max}/ = \mathit{max}/ \circ k+*$$

(In (62) and (63) $k$ denotes a number.)

**Proofs**

In each case we have to demonstrate that the definition of promotability is satisfied. For instance, to prove (58) we observe that

$$\mathbf{not}.(p \mathbf{\ and\ } q) = (\mathbf{not}.p) \mathbf{\ or\ } (\mathbf{not}.q)$$

and
$$\mathbf{not}.1_{\mathbf{and}} = \mathbf{not}.\mathbf{T} = \mathbf{F} = 1_{\mathbf{or}}$$

Equation (59) is similar. For the remainder it is necessary to know that, by definition, $1_{min} = \infty$, $1_{max} = -.\infty$, $k + 1_{min} = 1_{min}$ and $k + 1_{max} = 1_{max}$.
(End of proofs)

Properties (58) and (59) would normally be expressed as

$$\neg\forall(i :: f.i) = \exists(i :: \neg(f.i))$$

and
$$\neg\exists(i :: f.i) = \forall(i :: \neg(f.i))$$

Their equivalents in the Bird-Meertens formalism are

(64) $$\mathbf{not} \circ \mathbf{and}/ \circ f* = \mathbf{or}/ \circ (\mathbf{not} \circ f)*$$

(65) $$\mathbf{not} \circ \mathbf{or}/ \circ f* = \mathbf{and}/ \circ (\mathbf{not} \circ f)*$$

Equation (64), for example, is easily derived from (58) as follows.

$$
\begin{array}{ll}
 & \mathbf{not} \circ \mathbf{and}/ \circ f* \\
= & \qquad \{ (58) \} \\
 & \mathbf{or}/ \circ \mathbf{not}* \circ f* \\
= & \qquad \{ \text{map distributivity (33)} \} \\
 & \mathbf{or}/ \circ (\mathbf{not} \circ f)*
\end{array}
$$

If a function is $\oplus{\rightarrow}\oplus$-promotable we will say that it is $\oplus$-promotable. Similarly we use the term $\oplus$-distributive for $\oplus{\rightarrow}\oplus$-distributive.

In [1] $\oplus$-promotability was called generalised distributivity and $\oplus{\rightarrow}\otimes$-promotability was called generalised skew distributivity. The statement of the promotability theorem took the form:

Generalised Distributivity
If $f$ is $\oplus{\rightarrow}\otimes$-promotable then

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

The direct analogue of this in the Bird-Meertens formalism is the following.

$$f \circ \oplus/ \circ g* \circ p\triangleleft \;=\; \otimes/ \circ (f \circ g)* \circ p\triangleleft$$

This is easily derived from the promotability of $f$ as we did just now in the particular case of equation (64). Note, however, the conciseness in the Bird-Meertens formalism: the promotability theorem has a very compact form, its use is separated from the use of map-distributivity and the irrelevant "$p\triangleleft$" disappears.

# 9 Promotability Properties of Map, Filter and Reduce

In this section we return to the consideration of map, reduce and filter in order to examine their promotability properties. We begin with two useful lemmas.

**Lemma 66** *If $f$ is $\oplus{\rightarrow}\otimes$-promotable and $g$ is $\otimes{\rightarrow}\odot$-promotable then $g \circ f$ is $\oplus{\rightarrow}\odot$-promotable.*

**Lemma 67** *Suppose $f \in B \longrightarrow C$, $\oplus \in B \times B \longrightarrow B$, $\otimes \in C \times C \longrightarrow C$ and $t \in A \longrightarrow B$. Then if $f$ is $\oplus \!\to\! \otimes$-promotable we have*

$$f \circ \oplus/ \circ t* = \otimes/ \circ (f \circ t)*$$

Comparing the definition of promotability with the empty and join rules (44)-(46) and (50)-(52) for each of the three operators immediately yields the following.
For $f \in A \longrightarrow B$, $\oplus \in B \times B \longrightarrow B$, $p \in A \longrightarrow Bool$ we have:

(68) $\quad \oplus/ \circ +\!\!\!+/ = \oplus/ \circ \oplus/* \in Tree.(Tree.B) \longrightarrow B$

(69) $\quad f* \circ +\!\!\!+/ = +\!\!\!+/ \circ f** \in Tree.(Tree.A) \longrightarrow Tree.B$

(70) $\quad p\lhd \circ +\!\!\!+/ = +\!\!\!+/ \circ p\lhd* \in Tree.(Tree.A) \longrightarrow Tree.A$

Also, applying lemma (66) to the last three equations we obtain the following.

(71) $$f* \circ p\lhd \circ +\!\!\!+/ = +\!\!\!+/ \circ (f* \circ p\lhd)* \in Tree.(Tree.A) \longrightarrow Tree.B$$

(72) $$\oplus/ \circ f* \circ +\!\!\!+/ = \oplus/ \circ (\oplus/ \circ f*)* \in Tree.(Tree.A) \longrightarrow B$$

(73) $$\oplus/ \circ f* \circ p\lhd \circ +\!\!\!+/ = \oplus/ \circ (\oplus/ \circ f* \circ p\lhd)* \in Tree.(Tree.A) \longrightarrow B$$

An application of the above is the following theorem due to Bird [7].

**Theorem 74 (Bird's homomorphism lemma)** *The function $h$ of type $Tree.A \longrightarrow B$ is $+\!\!\!+ \!\to\! \oplus$-promotable if and only if $h = \oplus/ \circ (h \circ \tau)* \in Tree.A \longrightarrow B$.*

**Proof**
We note that

$$\begin{array}{ll} & h \\ = & \quad \{ \text{ unit (8) } \} \\ & h \circ 1_\circ \\ = & \quad \{ \text{ tree identity (37) } \} \\ & h \circ +\!\!\!+/ \circ \tau* \end{array}$$

Thus,

$$\begin{array}{ll} & h \text{ is } +\!\!+\!\!\rightarrow\!\oplus\text{-promotable} \\ \Rrightarrow & \qquad \{ \text{ lemma (67) } \} \\ & h \ = \ \oplus/ \circ (h \circ \tau)* \end{array}$$

Moreover,

$$\begin{array}{ll} & h \ = \ \oplus/ \circ (h \circ \tau)* \\ \Rrightarrow & \qquad \{ \text{ (72) with } f := h \circ \tau \ \} \\ & h \circ +\!\!+/ \ = \ \oplus/ \circ h* \\ \equiv & \qquad \{ \text{ promotability theorem (57) } \} \\ & h \text{ is } +\!\!+\!\!\rightarrow\!\oplus\text{-promotable} \end{array}$$

(End of proof)

**Corollary 75** *Bird's homomorphism lemma is also valid when "Tree" is replaced everywhere by "List", when "Tree" is replaced everywhere by "Bag", and when "Tree" is replaced everywhere by "Set".*

**Proof**

By repeating the above proof replacing the reference to (37) by reference to the relevant parts of (38).

(End of proof)

The type information in theorem 74 and corollary 75 is vital. For example, the function *depth* that computes the maximum depth of a tree (i.e. the maximum depth of embeddings of the join operator) is $+\!\!+\!\!\rightarrow\!\oplus$-promotable where $\oplus$ is defined by

$$m \oplus n \ = \ (m+1) max (n+1)$$

But $\oplus$ is not associative and so *depth* is not defined on lists. Similar promotability properties hold for the function mapping a list into the set of all its prefixes and for the size function on bags, but in each case the property is not valid at a higher level in the type hierarchy (although it is valid at lower levels).

A corollary of the homomorphism lemma anticipated in the proof of corollary 36 is the following.

(76) $$\oplus/ \ = \ \oplus/ \circ 1_{\circ}*$$

**Proof**

$$
\begin{array}{ll}
& \oplus/ \\
= & \qquad\qquad \{\ (68),\ \text{homomorphism lemma}\ (74)\ \} \\
& \oplus/ \circ (\oplus/ \circ \tau)* \\
= & \qquad\qquad \{\ \text{one-pt. rule}\ (48)\ \} \\
& \oplus/ \circ 1_{\circ}*
\end{array}
$$

(End of proof)

We remarked earlier that the range translation rule [1] appears to be difficult to formulate in the Bird-Meertens formalism. We conclude this section, however, with a property that approximates the range translation rule and that can be used to give non-inductive proofs of properties like $\sum(i : 0 \leq i < n : i) = n(n+1)/2$. In its most basic form the property is as follows.

(77) $$p\triangleleft \circ t* = t* \circ (p \circ t)\triangleleft$$

whenever $p \in A \longrightarrow \mathit{Bool}$ and $t \in C \longrightarrow A$.

**Proof**

$$
\begin{array}{ll}
 & p\triangleleft \ \circ \ t* \\
= & \qquad \{ \text{ filter (41) } \} \\
 & +\!\!\!+/ \ \circ \ (\tau\langle p\rangle 1_{+\!\!\!+^\wedge})* \ \circ \ t* \\
= & \qquad \{ \text{ map distributivity (33) } \} \\
 & +\!\!\!+/ \ \circ \ ((\tau\langle p\rangle 1_{+\!\!\!+^\wedge}) \ \circ \ t)* \\
= & \qquad \{ \text{ conditionals (32) } \} \\
 & +\!\!\!+/ \ \circ \ (\tau \circ t \ \langle p \circ t\rangle \ 1_{+\!\!\!+^\wedge} \circ t)* \\
= & \qquad \{ \text{ constants (10), (15) } \} \\
 & +\!\!\!+/ \ \circ \ (\tau \circ t \ \langle p \circ t\rangle \ 1_{+\!\!\!+^\wedge})* \\
= & \qquad \{ \text{ one-pt. rule (47), empty rule (44) } \} \\
 & +\!\!\!+/ \ \circ \ (t* \circ \tau \ \langle p \circ t\rangle \ t* \circ 1_{+\!\!\!+^\wedge})* \\
= & \qquad \{ \text{ conditionals (31) } \} \\
 & +\!\!\!+/ \ \circ \ (t* \ \circ \ (\tau \ \langle p \circ t\rangle \ 1_{+\!\!\!+^\wedge}))* \\
= & \qquad \{ \text{ map distributivity (33) } \} \\
 & +\!\!\!+/ \ \circ \ t** \ \circ \ (\tau \ \langle p \circ t\rangle \ 1_{+\!\!\!+^\wedge})* \\
= & \qquad \{ \text{ map promotability (69) } \} \\
 & t* \ \circ \ +\!\!\!+/ \ \circ \ (\tau \ \langle p \circ t\rangle \ 1_{+\!\!\!+^\wedge})* \\
= & \qquad \{ \text{ filter (41) } \} \\
 & t* \ \circ \ (p \circ t)\triangleleft
\end{array}
$$

(End of proof)

A straightforward corollary of (77) is obtained by taking the composition of the left and right sides with $\oplus/ \ \circ \ f*$ and then applying map distributivity.

(78) $$\oplus/ \ \circ \ f* \ \circ \ p\triangleleft \ \circ \ t* \ = \ \oplus/ \ \circ \ (f \ \circ \ t)* \ \circ \ (p \ \circ \ t)\triangleleft$$

whenever $t \in C \longrightarrow A, p \in A \longrightarrow Bool, f \in A \longrightarrow B$ and $\oplus \in B \times B \longrightarrow B$.

In the quantifier notation (78) would be expressed as

$$\oplus(i \in t.C : p.i : f.i) \ = \ \oplus(j \in C : p.(t.j) : f.(t.j))$$

For the record we append one last property. Although very elegant I regret to say that I have yet to find a practical application for it!

(79) $$* \ \circ \ (\circ/) \ = \ (\circ/) \ \circ \ (**)$$

**Proof**

By applying the definition of promotability one shows that map is $\circ$-promotable. Map distributivity (33) establishes that it is $\circ$-distributive. We leave the rest to the reader. One must show that $*.1_\circ = 1_\circ$, i.e., by currying (4), $1_\circ * = 1_\circ$.

(End of proof)

# 10 Trading and Identity Laws

We return to the more direct consideration of laws analogous to the quantifier rules by deriving so-called "trading" laws and the identity law. The terminology "trading" is E.W. Dijkstra's [13]; the laws govern the interchange of expressions between the range and function part of a quantification.

There is no direct connection between trading and the identity law (except that they are both instances of promotability). There is thus no real reason for grouping them together in the one section.

The trading rules are in fact instances of the following lemma.

**Lemma 80 (Trading)** $\oplus/ \circ f* \circ p\triangleleft = \oplus/ \circ (f\langle p\rangle 1_{\oplus^\wedge})*$

**Proof**

$$
\begin{array}{ll}
 & \oplus/ \circ f* \circ p\triangleleft \\
= & \qquad \{ \text{ filter (41) } \} \\
 & \oplus/ \circ f* \circ +\!\!+/ \circ (\tau\langle p\rangle 1_{+\!+^\wedge})* \\
= & \qquad \{ \text{ promotability (72), (67) } \} \\
 & \oplus/ \circ (\oplus/ \circ f* \circ (\tau\langle p\rangle 1_{+\!+^\wedge}))* \\
= & \qquad \{ \text{ conditionals (31) } \} \\
 & \oplus/ \circ (\oplus/ \circ f* \circ \tau\ \langle p\rangle\ \oplus/ \circ f* \circ 1_{+\!+^\wedge})* \\
= & \qquad \{ \text{ one-pt. and empty rules, (44), (45), (47), (48) } \} \\
 & \oplus/ \circ (f\langle p\rangle 1_{\oplus^\wedge})*
\end{array}
$$

(End of proof)

The well-known trading rules for universal and existential quantification are immediate corollaries of (80).

(81) $\quad \textbf{and}/ \circ q* \circ p\triangleleft = \textbf{and}/ \circ (p \Rightarrow q)* \in \mathit{Tree}.A \longrightarrow \mathit{Bool}$

(assuming $p, q \in A \longrightarrow Bool$) where $p \Rightarrow q$ is defined by $p \Rightarrow q \;=\; (\neg \circ p) \vee q$

$$(82) \qquad \mathbf{or}/ \circ q* \circ p\triangleleft \;=\; \mathbf{or}/ \circ (p \wedge q)*$$

To derive (81) and (82) from (80) it suffices to observe that $q\langle p\rangle 1_\wedge = q\langle p\rangle\mathbf{true} = p \Rightarrow q$ and $q\langle p\rangle 1_\vee = q\langle p\rangle\mathbf{false} = p \wedge q$, which identities are easily proved by extensionality. Less familiar consequences of (80) are the trading rules for equivalence and inequivalence.

$$(83) \qquad \equiv/ \circ q* \circ p\triangleleft \;=\; \equiv/ \circ (p \Rightarrow q)*$$

$$(84) \qquad \not\equiv/ \circ q* \circ p\triangleleft \;=\; \not\equiv/ \circ (p \wedge q)*$$

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

The literal translation of (81) and (82) into the quantifier notation yields

$$\forall(i : p.i : q.i) \;=\; \forall(i :: p.i \;\mathbf{implies}\; q.i)$$

and

$$\exists(i : p.i : q.i) \;=\; \exists(i :: p.i \;\mathbf{and}\; q.i)$$

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

$$\forall(i : r.i \;\mathbf{and}\; p.i : q.i) \;=\; \forall(i : r.i : p.i \;\mathbf{implies}\; q.i)$$

and

$$\exists(i : r.i \;\mathbf{and}\; p.i : q.i) \;=\; \exists(i : r.i : p.i \;\mathbf{and}\; q.i)$$

To derive the equivalent of these in the Bird-Meertens formalism it is necessary to interpose a lemma.

$$(85) \qquad p\triangleleft \circ q\triangleleft \;=\; (q \wedge p)\triangleleft$$

**Proof**

$$
\begin{array}{ll}
 & p\lhd \ \circ \ q\lhd \\
= & \qquad \{ \text{ filter (41) } \} \\
 & +\!\!\!+/ \ \circ \ (\tau\langle p\rangle 1_{+\!\!+^\wedge})* \ \circ \ +\!\!\!+/ \ \circ \ (\tau\langle q\rangle 1_{+\!\!+^\wedge})* \\
= & \qquad \{ \text{ promotability (72), (67) } \} \\
 & +\!\!\!+/ \ \circ \ (+\!\!\!+/ \ \circ \ (\tau\langle p\rangle 1_{+\!\!+^\wedge})* \ \circ \ (\tau\langle q\rangle 1_{+\!\!+^\wedge}))* \\
= & \qquad \{ \text{ conditionals (31) } \} \\
 & +\!\!\!+/ \ \circ \ (+\!\!\!+/ \circ (\tau\langle p\rangle 1_{+\!\!+^\wedge})* \circ \tau \ \langle q\rangle \ +\!\!\!+/ \circ (\tau\langle p\rangle 1_{+\!\!+^\wedge})* \circ 1_{+\!\!+^\wedge})* \\
= & \qquad \{ \text{ one-pt. and empty rules (44), (45), (47), (48) } \} \\
 & +\!\!\!+/ \ \circ \ ((\tau\langle p\rangle 1_{+\!\!+^\wedge}) \ \langle q\rangle \ 1_{+\!\!+^\wedge})* \\
= & \qquad \{ \text{ conditionals (29) } \} \\
 & +\!\!\!+/ \ \circ \ (\tau \ \langle p \wedge q\rangle \ 1_{+\!\!+^\wedge})* \\
= & \qquad \{ \text{ filter (41) } \} \\
 & (p \wedge q)\lhd
\end{array}
$$

(End of proof)

We leave it to the reader to verify that the quantifier rules are a straightforward combination of (85) with (81) and (82).

Sometimes the importance of simple properties is overlooked because of their apparent triviality. The following lemma is an instance.

**Lemma 86** *For arbitrary operator $\oplus$ the function $1_{\oplus^\wedge}$ is $\oplus$-promotable. I.e.*

$$1_{\oplus^\wedge} \ \circ \ \oplus/ \ = \ \oplus/ \ \circ \ (1_{\oplus^\wedge})*$$

**Proof**

We verify that $1_{\oplus^\wedge}$ satisfies the definition of $\oplus$-promotability. First, $1_{\oplus^\wedge}$ is $\oplus$-distributive.

$$
\begin{array}{ll}
 & 1_{\oplus^\wedge}.(x \oplus y) \\
= & \qquad \{ \text{ lifting (15), constants (9) } \} \\
 & 1_\oplus \\
= & \qquad \{ \text{ unit (7) } \} \\
 & 1_\oplus \oplus 1_\oplus \\
= & \qquad \{ \text{ lifting (15), constants (9) } \} \\
 & 1_{\oplus^\wedge}.x \oplus 1_{\oplus^\wedge}.y
\end{array}
$$

Second, $1_{\oplus^\wedge}$ preserves the unit of $\oplus$.

$$
\begin{array}{cl}
 & 1_{\oplus} \\
= & \qquad \{ \text{ lifting (15), constants (9) } \} \\
 & 1_{\oplus^\wedge}.1_{\oplus}
\end{array}
$$

(End of proof)

An immediate corollary of (86), obtained by applying properties (15) and (10) of lifting, is the following.

(87) $$1_{\oplus^\wedge} \;=\; \oplus/ \;\circ\; (1_{\oplus^\wedge})*$$

Hardly surprising! But from (87) we can write down a formula for **false**$\triangleleft$.

(88) $$\mathbf{false}\triangleleft \;=\; 1_{+\!\!+^\wedge}$$

**Proof**

$$
\begin{array}{cl}
 & \mathbf{false}\triangleleft \\
= & \qquad \{ \text{ filter (41) } \} \\
 & +\!\!+/ \;\circ\; (\tau \,\langle \mathbf{false} \rangle\; 1_{+\!\!+^\wedge})* \\
= & \qquad \{ \text{ conditionals (27) } \} \\
 & +\!\!+/ \;\circ\; (1_{+\!\!+^\wedge})* \\
= & \qquad \{ \text{ (87) } \} \\
 & 1_{+\!\!+^\wedge}
\end{array}
$$

(End of proof)

A second consequence is the rule called the **identity rule** in [1].

(89) $$\oplus/ \;\circ\; f* \;\circ\; \mathbf{false}\triangleleft \;=\; 1_{\oplus^\wedge}$$

**Proof**

$$
\begin{array}{cl}
 & \oplus/ \;\circ\; f* \;\circ\; \mathbf{false}\triangleleft \\
= & \qquad \{ \text{ (88) } \} \\
 & \oplus/ \;\circ\; f* \;\circ\; 1_{+\!\!+^\wedge} \\
= & \qquad \{ \text{ empty rules (44), (45) } \} \\
 & 1_{\oplus^\wedge}
\end{array}
$$

(End of proof)

Examples of (89) are $+/ \circ f* \circ \mathbf{false}\triangleleft = 0\bullet$, $\times/ \circ f* \circ \mathbf{false}\triangleleft = 1\bullet$, $\mathbf{and}/ \circ f* \circ \mathbf{false}\triangleleft = \mathbf{true}$, and $\mathbf{or}/ \circ f* \circ \mathbf{false}\triangleleft = \mathbf{false}$. These are written in the quantifier form as follows.

$$\begin{array}{rcl} \sum(i : \mathbf{false} : f.i) & = & 0 \\ \prod(i : \mathbf{false} : f.i) & = & 1 \\ \forall(i : \mathbf{false} : f.i) & = & \mathbf{true} \\ \exists(i : \mathbf{false} : f.i) & = & \mathbf{false} \end{array}$$

Note that the proof here is valid for trees (and thus also for lists, bags and sets provided $\oplus$ has the appropriate properties). This is an improvement on the one in [1] since there the range splitting rule (see section 12) was used. In order to use range splitting it is necessary to assume that $\oplus$ is associative and symmetric. In other words the proof in [1] is valid for bags (and hence also for sets) but not for trees or lists.

# 11 Lifting Revisited

A whole host of new properties can be discovered by considering promotability in relation to the lifting operator. Basically, lifting preserves promotability properties. More precisely we have:

**Theorem 90 (Lifting theorem)** *If $f$ is $\oplus{\rightarrow}\otimes$-promotable then $(f\circ)$ is $\hat{\oplus}{\rightarrow}\hat{\otimes}$-promotable.*

**Proof**

By application of the definition of promotability. First, note that

$$\begin{array}{ll} & f.(g.x \oplus h.x) \;=\; f.(g.x) \otimes f.(h.x) \\ \equiv & \qquad \{ \text{ functions (2), lifting (13) } \} \\ & (f \circ (g \hat{\oplus} h)).x \;=\; ((f \circ g) \hat{\otimes} (f \circ h)).x \end{array}$$

Thus,

$$\begin{array}{ll} & f \text{ is } \oplus{\rightarrow}\otimes\text{-distributive} \\ \Rrightarrow & \qquad \{ \text{ extensionality } \} \\ & f \circ (g \hat{\oplus} h) \;=\; (f \circ g) \hat{\otimes} (f \circ h) \\ \equiv & \qquad \{ \text{ currying (5) } \} \\ & (f\circ).(g \hat{\oplus} h) \;=\; ((f\circ).g) \hat{\otimes} ((f\circ).h) \\ \equiv & \qquad \{ \text{ definition (55) } \} \\ & (f\circ) \text{ is } \hat{\oplus}{\rightarrow}\hat{\otimes}\text{-distributive} \end{array}$$

Also,

$$
\begin{array}{ll}
 & (f\circ).1_{\oplus^\wedge} \\
= & \qquad \{ \text{ currying (5), lifting (15) } \} \\
 & f \;\circ\; (1_{\oplus})_\bullet \\
= & \qquad \{ \text{ lifting (11) } \} \\
 & (f.1_{\oplus})_\bullet \\
= & \qquad \{ \; f \text{ is } \oplus{\rightarrow}\otimes\text{-promotable } \} \\
 & (1_{\otimes})_\bullet \\
= & \qquad \{ \text{ lifting (15) } \} \\
 & 1_{\oplus^\wedge}
\end{array}
$$

The theorem follows by application of the promotability theorem.
(End of proof)

Elementary examples of the lifting theorem are obtained by applying the definitions of $\neg$, $\wedge$ and $\vee$ to (58) and (59).

(91) $$\neg \circ \wedge\!\!\!/ \;=\; \vee\!/ \circ \neg *$$

(92) $$\neg \circ \vee\!/ \;=\; \wedge\!/ \circ \neg *$$

By use of the lifting theorem we could now repeat all of the corollaries of the promotability theorem in a "lifted" form. Such corollaries can also be "lifted" (since if $f$ is $\oplus{\rightarrow}\otimes$-distributive by two applications of the lifting theorem we get that $(f\circ\circ)$ is $\oplus^{\wedge\wedge}{\rightarrow}\otimes^{\wedge\wedge}$-distributive) and the process can be repeated ad infinitum. In order not to overwhelm the reader we limit ourselves to the statement of those properties that we will have occasion to use later.

First $(f*\circ)$ and $(p\triangleleft\circ)$ are $+\!\!\!+^\wedge$-promotable.

(93) $$(f*\circ) \;\circ\; +\!\!\!+^\wedge/ \;=\; +\!\!\!+^\wedge/ \;\circ\; f*\circ*$$

(94) $$(p\triangleleft\circ) \;\circ\; +\!\!\!+^\wedge/ \;=\; +\!\!\!+^\wedge/ \;\circ\; p\triangleleft\circ*$$

Also, $(\oplus/\circ)$ is $+\!\!\!+^\wedge{\rightarrow}\oplus^\wedge$-promotable.

(95) $$(\oplus/\circ) \;\circ\; (+\!\!\!+^\wedge/) \;=\; (\oplus^\wedge/) \;\circ\; \oplus/\circ*$$

In the above we assume that $\oplus \in B{\times}B \longrightarrow B$, $f \in A{\longrightarrow}B$ and $p \in A{\longrightarrow}Bool$. Finally we note that, for arbitrary $h \in C{\longrightarrow}D$, we have $(h\tilde{\circ})$ is $\oplus^\wedge$-promotable. This is because, by lifting (14) and currying (6),

$$(h\tilde{\circ}).(f \oplus^\wedge g) \;=\; (h\tilde{\circ}).f \oplus^\wedge (h\tilde{\circ}).g$$

and, by properties (10) and (16) of constants and currying (6),

$$(h\tilde{\circ}).1_{\oplus^\wedge} \;=\; 1_{\oplus^\wedge}$$

Thus by the promotability theorem we have:

(96) $$(h\tilde{\circ}) \circ \oplus^\wedge/ \;=\; \oplus^\wedge/ \circ (h\tilde{\circ})*$$

# 12 Associativity and Symmetry

We now turn our attention to binary operators that are both associative and symmetric. We assume, therefore, throughout this section that $\oplus$ denotes such an operator. Since within bags the join operator is associative and symmetric particular cases of the properties we prove (when $\oplus$ is instantiated to $+\!\!+$) describe some of the essential properties of bags.

It is of particular importance throughout this section (and indeed the next) to note the supplied type information. In general we establish equalities between functions with *bags* as domains; such equalities are also valid when the domains are trees but the stated equalities are stronger.

To avoid repeating type assumptions we assume throughout that $f, g \in A{\longrightarrow}B$, $p, q \in A{\longrightarrow}Bool$ and $\oplus \in B{\times}B \longrightarrow B$. Other variables are introduced as we go along.

The most basic property arising from the assumption of associativity and symmetry is the following.

(97) $$\oplus/ \circ (f \oplus^\wedge g)* \;=\; (\oplus/ \circ f*) \oplus^\wedge (\oplus/ \circ g*)$$

**Proof**

The proof is by structural induction. First we have

$$
\begin{array}{ll}
 & (\oplus/ \circ (f \oplus^{\wedge} g)*).1_{++} \\
= & \qquad \{ \text{ empty rules (44), (45) } \} \\
 & 1_{\oplus} \\
= & \qquad \{ \text{ unit (7) } \} \\
 & 1_{\oplus} \oplus 1_{\oplus} \\
= & \qquad \{ \text{ empty rules (44), (45) } \} \\
 & (\oplus/ \circ f*).1_{++} \oplus (\oplus/ \circ g*).1_{++} \\
= & \qquad \{ \text{ lifting (13) } \} \\
 & ((\oplus/ \circ f*) \oplus^{\wedge} (\oplus/ \circ g*)).1_{++}
\end{array}
$$

Next,

$$
\begin{array}{ll}
 & \oplus/ \circ (f \oplus^{\wedge} g)* \circ \tau \\
= & \qquad \{ \text{ one-pt. rules (47), (48) } \} \\
 & f \oplus^{\wedge} g \\
= & \qquad \{ \text{ one-pt. rules (47), (48) } \} \\
 & (\oplus/ \circ f* \circ \tau) \oplus^{\wedge} (\oplus/ \circ g* \circ \tau) \\
= & \qquad \{ \text{ lifting (14) } \} \\
 & ((\oplus/ \circ f*) \oplus^{\wedge} (\oplus/ \circ g*)) \circ \tau
\end{array}
$$

Finally, the induction step proceeds as follows.

$$
\begin{array}{lll}
|[ & s, t \in \mathit{Bag}.A & \\
; & (\oplus/ \circ (f \oplus^{\wedge} g)*).s = ((\oplus/ \circ f*) \oplus^{\wedge} (\oplus/ \circ g*)).s & \\
; & (\oplus/ \circ (f \oplus^{\wedge} g)*).t = ((\oplus/ \circ f*) \oplus^{\wedge} (\oplus/ \circ g*)).t & \\
\triangleright & & (\oplus/ \circ (f \oplus^{\wedge} g)*).(s \mathbin{+\!\!+} t) \\
 & = & \qquad \{ \text{ join rules (50), (51) } \} \\
 & & (\oplus/ \circ (f \oplus^{\wedge} g)*).s \oplus (\oplus/ \circ (f \oplus^{\wedge} g)*).t \\
 & = & \qquad \{ \text{ inductive hypotheses } \} \\
 & & ((\oplus/ \circ f*) \oplus^{\wedge} (\oplus/ \circ g*)).s \oplus ((\oplus/ \circ f*) \oplus^{\wedge} (\oplus/ \circ g*)).t \\
 & = & \qquad \{ \text{ lifting (13) } \} \\
 & & ((\oplus/ \circ f*).s \oplus (\oplus/ \circ g*).s) \oplus ((\oplus/ \circ f*).t \oplus (\oplus/ \circ g*).t) \\
 & = & \qquad \{ \text{ associativity and symmetry of } \oplus \} \\
 & & ((\oplus/ \circ f*).s \oplus (\oplus/ \circ f*).t) \oplus ((\oplus/ \circ g*).s \oplus (\oplus/ \circ g*).t) \\
 & = & \qquad \{ \text{ join rules (50), (51) } \} \\
 & & (\oplus/ \circ f*).(s \mathbin{+\!\!+} t) \oplus (\oplus/ \circ g*).(s \mathbin{+\!\!+} t) \\
 & = & \qquad \{ \text{ lifting (13) } \} \\
 & & ((\oplus/ \circ f*) \oplus^{\wedge} (\oplus/ \circ g*)).(s \mathbin{+\!\!+} t) \\
]| & &
\end{array}
$$

(End of proof)

With the quantifier calculus in mind let us extend the left side of (97) with a filter component. The right side must also be extended; a good guess as to how this is accomplished gives the following.

**Theorem 98 (Associativity and Symmetry)** *For associative and symmetric operator* $\oplus$ *we have*

$$\oplus/ \circ (f \oplus\hat{} \, g)* \circ p\triangleleft = (\oplus/ \circ f* \circ p\triangleleft) \oplus\hat{} \, (\oplus/ \circ g* \circ p\triangleleft)$$

Theorem (98) is the direct analogue of the associativity and symmetry rule in [1] expressed in the quantifier calculus as follows.

$$\oplus(i \in A : p.i : f.i \oplus g.i) = \oplus(i \in A : p.i : f.i) \oplus \oplus(i \in A : p.i : g.i)$$

**Proof**

$$\begin{array}{ll} & \oplus/ \circ (f \oplus\hat{} \, g)* \circ p\triangleleft \\ = & \qquad \{ \ (97) \ \} \\ & ((\oplus/ \circ f*) \oplus\hat{} \, (\oplus/ \circ g*)) \circ p\triangleleft \\ = & \qquad \{ \text{ lifting } (14) \ \} \\ & (\oplus/ \circ f* \circ p\triangleleft) \oplus\hat{} \, (\oplus/ \circ g* \circ p\triangleleft) \end{array}$$

(End of proof)

A fundamental property of bags that follows almost immediately from (97) is the following version of range splitting [1]. Pay particular attention to the type information!

**Range Splitting**

(99) $$(p\triangleleft) +\!\!+\hat{} \, (\neg.p)\triangleleft = 1_\circ \in Bag.A \longrightarrow Bag.A$$

**Proof**

$$
\begin{array}{ll}
 & (p\triangleleft) +\!\!+^{\wedge} (\neg.p)\triangleleft \\
= & \qquad \{ \text{ filter (41) } \} \\
 & (+\!\!+/ \circ (\tau\langle p\rangle 1_{+\!+^{\wedge}})*) +\!\!+^{\wedge} (+\!\!+/ \circ (\tau\langle \neg.p\rangle 1_{+\!+^{\wedge}})*) \\
= & \qquad \{ \text{ (97), } +\!\!+ \text{ is associative and symmetric in } Bag.A \text{ } \} \\
 & +\!\!+/ \circ ((\tau\langle p\rangle 1_{+\!+^{\wedge}}) +\!\!+^{\wedge} (\tau\langle \neg.p\rangle 1_{+\!+^{\wedge}}))* \\
= & \qquad \{ \text{ case analysis and extensionality } \} \\
 & +\!\!+/ \circ \tau* \\
= & \qquad \{ \text{ tree-identity, (38) } \} \\
 & 1_{\circ}
\end{array}
$$

(End of proof)

As usual, we extend (99) to expressions of the form $\oplus/ \circ f* \circ p\triangleleft$ in order to obtain the exact analogue of the range splitting rule for quantifiers.

**Corollary 100** *For associative and symmetric* $\oplus$:

$$\oplus/ \circ f* \circ p\triangleleft \;=\; (\oplus/ \circ f* \circ (p \wedge q)\triangleleft) \oplus^{\wedge} (\oplus/ \circ f* \circ (p \wedge \neg.q)\triangleleft)$$

**Proof**

$$
\begin{array}{ll}
 & \oplus/ \circ f* \circ p\triangleleft \\
= & \qquad \{ \text{ (99) } \} \\
 & \oplus/ \circ f* \circ p\triangleleft \circ ((q\triangleleft) +\!\!+^{\wedge} (\neg.q)\triangleleft) \\
= & \qquad \{ \text{ lifting theorem (90), join rules (50), (51), (52) } \} \\
 & (\oplus/ \circ f* \circ p\triangleleft \circ q\triangleleft) \oplus^{\wedge} (\oplus/ \circ f* \circ p\triangleleft \circ (\neg.q)\triangleleft) \\
= & \qquad \{ \text{ trading rules (85) } \} \\
 & (\oplus/ \circ f* \circ (q \wedge p)\triangleleft) \oplus^{\wedge} (\oplus/ \circ f* \circ (\neg.q \wedge p)\triangleleft)
\end{array}
$$

(End of proof)

The forms of (97) and (98) both suggest distributivity properties. In the next few paragraphs we explore this aspect of the equations culminating in a version of the cartesian product rule of [1].

We can indeed verify that (97) is a distributivity property by observing that $\oplus/ \circ h* = ((\oplus/\circ) \circ *).h$ (by the currying rules (4) and (5)). Thus rewriting (97) we obtain the following.

(101) $\qquad ((\oplus/\circ) \circ *).(f \oplus^{\wedge} g) \;=\; ((\oplus/\circ) \circ *).f \oplus^{\wedge} ((\oplus/\circ) \circ *).g$

In other words, $(\oplus/\circ) \circ *$ is $\oplus^\wedge$-distributive. Does it preserve $1_{\oplus^\wedge}$? I.e. can we prove

(102) $$((\oplus/\circ) \circ *).1_{\oplus^\wedge} = 1_{\oplus^\wedge}$$

Indeed we can as follows

$$\begin{array}{ll} & ((\oplus/\circ) \circ *).1_{\oplus^\wedge} \\ = & \qquad \{ \text{ currying (5) } \} \\ & \oplus/ \circ (1_{\oplus^\wedge})* \\ = & \qquad \{ \text{ identity rules (87) } \} \\ & 1_{\oplus^\wedge} \end{array}$$

We have thus proved that $(\oplus/\circ) \circ *$ is $\oplus^\wedge$-promotable.

(103) $$(\oplus/\circ) \circ * \circ \oplus^\wedge/ = \oplus^\wedge/ \circ ((\oplus/\circ) \circ *)*$$

It is a little more difficult to identify (98) as a distributivity property but with practice the manipulations become routine. First we use currying (6) to rewrite $h \circ p\triangleleft$ as $(p\triangleleft\tilde{\circ}).h$. Also rewriting $\oplus/ \circ h*$ as $((\oplus/\circ) \circ *).h$, as before, (98) may be rewritten as follows.

(104) $$\begin{array}{l} ((p\triangleleft\tilde{\circ}) \circ (\oplus/\circ) \circ *).(f \oplus^\wedge g) \\ \qquad = ((p\triangleleft\tilde{\circ}) \circ (\oplus/\circ) \circ *).f \oplus^\wedge ((p\triangleleft\tilde{\circ}) \circ (\oplus/\circ) \circ *).g \end{array}$$

Thus (98) is indeed a distributivity property. In order to apply the promotability theorem we need to verify that

$$((p\triangleleft\tilde{\circ}) \circ (\oplus/\circ) \circ *).1_{\oplus^\wedge} = 1_{\oplus^\wedge}$$

but this easily follows from (102) and constants (10) and (15). We thus conclude by the promotability theorem

(105) $$\begin{array}{l} (p\triangleleft\tilde{\circ}) \circ (\oplus/\circ) \circ * \circ \oplus^\wedge/ \\ \qquad = \oplus^\wedge/ \circ ((p\triangleleft\tilde{\circ}) \circ (\oplus/\circ) \circ *)* \end{array}$$

Introducing the variable $u \in Bag.(A \longrightarrow B)$ and reversing the steps used to introduce the expressions "$(p\triangleleft\tilde{\circ})$" and "$(\oplus/\circ) \circ *$" we get in turn

$$((\oplus/\circ) \circ * \circ \oplus^\wedge/).u \circ p\triangleleft = (\oplus^\wedge/ \circ ((p\triangleleft\tilde{\circ}) \circ (\oplus/\circ) \circ *)*).u$$

and,

$$\oplus\!/ \circ (\oplus\hat{\ }/.u)* \circ p\triangleleft \;=\; ((\oplus\hat{\ }/) \circ ((p\triangleleft\tilde{\circ}) \circ (\oplus\!/\circ) \circ *)*).u$$

We now introduce variables $t \in Bag.C$, $r \in C \longrightarrow Bool$, $h \in C \longrightarrow A \longrightarrow B$ and make the substitution $(h* \circ r\triangleleft).t$ for $u$. The left side becomes

$$\oplus\!/ \circ ((\oplus\hat{\ }/ \circ h* \circ r\triangleleft).t)* \circ p\triangleleft$$

and the right side simplifies as follows

$$\begin{array}{ll}
 & (\oplus\hat{\ }/ \circ ((p\triangleleft\tilde{\circ}) \circ (\oplus\!/\circ) \circ *)*).((h* \circ r\triangleleft).t) \\
= & \qquad\{ \text{ functions (2) } \} \\
 & (\oplus\hat{\ }/ \circ ((p\triangleleft\tilde{\circ}) \circ (\oplus\!/\circ) \circ *)* \circ h* \circ r\triangleleft).t \\
= & \qquad\{ \text{ map distributivity (33), definition of } g \text{ } \} \\
 & (\oplus\hat{\ }/ \circ g* \circ r\triangleleft).t
\end{array}$$

where, for $x \in C$,

$$\begin{array}{ll}
 & g.x \\
= & \qquad\{ \text{ by definition } \} \\
 & ((p\triangleleft\tilde{\circ}) \circ (\oplus\!/\circ) \circ * \circ h).x \\
= & \qquad\{ \text{ currying (5) and (6), functions (2) } \} \\
 & \oplus\!/ \circ (h.x)* \circ p\triangleleft
\end{array}$$

We have thus proved the following theorem.

**Theorem 106 (Cartesian Product)** *For associative and symmetric* $\oplus \in B{\times}B \longrightarrow B$, $p \in A \longrightarrow Bool$, $t \in Bag.C$, $x \in C$, $h \in C \longrightarrow A \longrightarrow B$ *and* $r \in C \longrightarrow Bool$,

$$\oplus\!/ \circ ((\oplus\hat{\ }/ \circ h* \circ r\triangleleft).t)* \circ p\triangleleft = (\oplus\hat{\ }/ \circ g* \circ r\triangleleft).t$$

*where* $g.x = \oplus\!/ \circ (h.x)* \circ p\triangleleft$

Within the quantifier calculus the Cartesian Product rule looks much more familiar. Specifically, it is the following

$$\begin{array}{ll}
 & \oplus(y \in A : p.y : \oplus(x \in C : r.x : h.x.y)) \\
= & \oplus(x \in C : r.x : \oplus(y \in A : p.y : h.x.y))
\end{array}$$

(The notation used in the last formula has been chosen to facilitate comparison with theorem 106.)

# 13 Adding Idempotency

In this last section we consider properties arising from the addition of idempotency to the list of assumptions about the $\oplus$ operator. In particular we consider functions with range $Set.A$; in other words we assume that $+\!\!+$ is associative, symmetric and idempotent.

Two rules that are important consequences of the idempotency of $+\!\!+$ are the rule of range disjunction and its generalisation.

**Range Disjunction**

(107) $$(p \vee q)\lhd \;=\; p\lhd +\!\!+^{\hat{}}\, q\lhd \;\in\; Set.A \longrightarrow Set.A$$

(where $p, q \in A \longrightarrow Bool$).

**Proof**

$$
\begin{array}{ll}
& p\lhd +\!\!+^{\hat{}}\, q\lhd \\
= & \qquad \{ \text{ range splitting (99), type hierarchy } \} \\
& (p\lhd \circ (q\lhd +\!\!+^{\hat{}}\, (\neg.q)\lhd)) +\!\!+^{\hat{}}\, (q\lhd \circ (p\lhd +\!\!+^{\hat{}}\, (\neg.p)\lhd)) \\
= & \qquad \{ \text{ lifting theorem (90), join rule (52) } \} \\
& ((p\lhd \circ q\lhd) +\!\!+^{\hat{}}\, (p\lhd \circ (\neg.q)\lhd)) +\!\!+^{\hat{}}\, ((q\lhd \circ p\lhd) +\!\!+^{\hat{}}\, (q\lhd \circ (\neg.p)\lhd)) \\
= & \qquad \{ +\!\!+^{\hat{}} \text{ is associative and symmetric in } Set.A \longrightarrow Set.A \} \\
& (p\lhd \circ q\lhd) +\!\!+^{\hat{}}\, (q\lhd \circ p\lhd) +\!\!+^{\hat{}}\, (p\lhd \circ (\neg.q)\lhd) +\!\!+^{\hat{}}\, (q\lhd \circ (\neg.p)\lhd) \\
= & \qquad \{ \text{ trading (85), } +\!\!+^{\hat{}} \text{ is idempotent in } Set.A \longrightarrow Set.A \} \\
& (p\lhd \circ q\lhd) +\!\!+^{\hat{}}\, (p\lhd \circ (\neg.q)\lhd) +\!\!+^{\hat{}}\, (q\lhd \circ (\neg.p)\lhd) \\
= & \qquad \{ \text{ lifting theorem (90), join rule (52) } \} \\
& (p\lhd \circ (q\lhd +\!\!+^{\hat{}}\, (\neg.q)\lhd)) +\!\!+^{\hat{}}\, (q\lhd \circ (\neg.p)\lhd) \\
= & \qquad \{ \text{ range splitting (99), type hierarchy } \} \\
& (p\lhd \circ 1_\circ) +\!\!+^{\hat{}}\, (q\lhd \circ (\neg.p)\lhd) \\
= & \qquad \{ \text{ unit (8), predicate calculus, trading (85) } \} \\
& (p \wedge (p \vee q))\lhd +\!\!+^{\hat{}}\, ((\neg.p) \wedge (p \vee q))\lhd \\
= & \qquad \{ \text{ trading (85) } \} \\
& ((p \vee q)\lhd \circ p\lhd) +\!\!+^{\hat{}}\, ((p \vee q)\lhd \circ (\neg.p)\lhd) \\
= & \qquad \{ \text{ lifting theorem (90), join rule (52) } \} \\
& (p \vee q)\lhd \circ (p\lhd +\!\!+^{\hat{}}\, (\neg.p)\lhd) \\
= & \qquad \{ \text{ range splitting (99), type hierarchy } \} \\
& (p \vee q)\lhd
\end{array}
$$

(End of proof)

**Corollary 108** *For associative, symmetric and idempotent* $\oplus$,

$$\oplus/ \circ f* \circ (p \vee q)\triangleleft \;=\; (\oplus/ \circ f* \circ p\triangleleft) \oplus^{\wedge} (\oplus/ \circ f* \circ q\triangleleft)$$

Equation (108) is the direct equivalent of the range disjunction rule given in [1]

The range disjunction rule is clearly recognisable as a distributivity property. Now

$$\begin{array}{lll} & 1_{\vee}\triangleleft & \\ = & & \{\text{ predicate calculus }\} \\ & \mathbf{false}\triangleleft & \\ = & & \{\text{ identity rules (88) }\} \\ & 1_{+\!\!+^{\wedge}} & \end{array}$$

Thus, applying the promotability theorem (theorem 57), we have the following generalisation of range disjunction.

**Generalised Range Disjunction**

(109) $\quad \triangleleft \circ \vee/ \;=\; +\!\!+^{\wedge}/ \circ (\triangleleft *) \;\in\; [\![Set.(\alpha{\longrightarrow}Bool){\longrightarrow}Set.\alpha{\longrightarrow}Set.\alpha]\!]$

The form of the generalised range disjunction rule in the quantifier calculus is as follows.

$$\begin{array}{ll} & \oplus(y \in A : \exists(x \in C : r.x : h.x.y) : f.y) \\ = & \oplus(x \in C : r.x : \oplus(y \in A : h.x.y : f.y)) \end{array}$$

We obtain its equivalent in the Bird-Meertens formalism in much the same way that we obtained the cartesian product rule (theorem 106). Here then is its statement. Readers who have reached this point in the text are rewarded by the pleasure of constructing the proof for themselves!

**Theorem 110** *Suppose* $\oplus \in B{\times}B \longrightarrow B$ *is associative, symmetric and idempotent,* $r \in C{\longrightarrow}Bool$, $h \in C{\longrightarrow}A{\longrightarrow}Bool$, $x \in C$, $f \in A{\longrightarrow}B$ *and* $t \in Set.C$. *Then*

$$\oplus/ \circ f* \circ ((\vee/ \circ h* \circ r\triangleleft).t)\triangleleft \;=\; (\oplus^{\wedge}/ \circ g* \circ r\triangleleft).t$$

*where* $g.x = \oplus/ \circ f* \circ (h.x)\triangleleft$

# 14 Conclusions

When originally setting out on this paper I had planned to call it "a comparison of the Eindhoven quantifier notation and the Bird-Meertens formalism". Now, with over a hundred equations behind me but nary a program in sight, it seems difficult to make any comparatory remarks. I shall try nevertheless.

The paper has shown that the basic properties of quantification over finite domains can all be expressed and proved within the Bird-Meertens formalism. However, thence to conclude that the expressiveness of the formalisms is to some extent equivalent would be quite wrong.

The Bird-Meertens formalism is apparently more economical than the quantifier notation and permits a much greater separation of concerns in the statement and proof of the basic properties. We have seen several such properties whose statement in the Bird-Meertens formalism is extraordinarily compact and devoid of irrelevant information whereas the corresponding property in the quantifier calculus is quite the opposite. (Compare for example the identity rules (88) and (89), the range splitting rules (99) and (100), and the range disjunction rules (107) and (108).)

On the other hand the statement of the cartesian product and generalised range disjunction rules in the Bird-Meertens formalism are disappointingly ugly; I cannot imagine anyone applying either of the rules as they have been formulated here. The lack of a clean cartesian product rule is a distinct drawback since the decision whether to rewrite $\oplus(i,j : r.i \wedge p.j : f.i.j)$ as $\oplus(i : r.i : \oplus(j : p.j : f.i.j))$ or $\oplus(j : p.j : \oplus(i : r.i : f.i.j))$ can be crucial in algorithm design. Because the step cannot be cleanly expressed it is one that is commonly disregarded in accounts of functional programs. (An example would be whether to express the segments of a list as the tail portions of the initial portions of the list or as the initial portions of the tail portions of the list. The commitment as to which to choose is often taken at the so-called "specification level".) Generalised range disjunction is a rule that is not commonly known and its use side-stepped, but that is a different story. For further discussion and an example of its use see the account of shortest path algorithms in [1].

Exploration of the cartesian product rule and generalised range disjunction has had the spin-off, however, of forcing me to examine lifting in much closer detail — resulting in the lifting theorem and, I believe, in much cleaner accounts of the more basic properties. These two rules have also tested the

Bird-Meertens formalism to a much greater degree, in my view, than any other calculations that have so far been published. Nevertheless, there is still room for improvement here. It is worth noting that Meertens [20] advocates a notation in which function application *and* function composition are denoted silently by juxtaposition. Such economies of notation have the disastrous effect of making theorems like the lifting theorem almost impossible to state. However it may be that lifting is such an important operator that a notation should indeed be devised that makes its use almost invisible. I don't know!

The first impression of the novice on reading a paper such as this will very likely be bewilderment at the immense number of formulae. How could one ever be expected to remember them all? The point is though that one has to remember very few. The avoidance of induction (equally recursion) as far as possible means that the formulae can be recovered, if necessary, by simple calculation. In principle, therefore, it suffices to record the basic definitions (function composition and application, lifting, conditionals, map, reduce and filter) and the four properties proved by induction (map distributivity, the tree identity theorem, the promotability theorem and the associativity and symmetry law). In practice, of course, one soon acquires a relatively large repertoire of more-commonly-used properties.

I used the word "exploration" in the title of the paper because that is exactly what it has been. There seem to be too many taboos about breaking with conventional or familiar notations that act as a barrier to this sort of exploration. It is only within the last few years that I, myself, have been stimulated to seriously think about and experiment with matters of notation (thanks largely to the work of van Gasteren and Dijkstra [22]), but the issue is too important for it to be ignored or to be discussed at the level of inculcated prejudice.

# APPENDIX A
## Summary of equations and theorems

Functions

$$\begin{array}{lrcl}
(1) & h.x.y & = & (h.x).y \\
(2) & (f \circ g).x & = & f.(g.x) \\
(3) & f \circ (g \circ h) & = & (f \circ g) \circ h
\end{array}$$

Currying

$$\begin{array}{lrcl}
(4) & x\odot & = & \odot.x \\
(5) & (f\circ).g & = & f \circ g \\
(6) & (f\breve{\circ}).g & = & g \circ f
\end{array}$$

Unit

$$\begin{array}{lrcl}
(7) & y & = & 1_{\oplus} \oplus y \\
(8) & y & = & y \oplus 1_{\oplus}
\end{array}$$

Constants and Lifting

$$\begin{array}{lrcl}
(9) & a\bullet.x & = & a \\
(10) & a\bullet \circ f & = & a\bullet \\
(11) & f \circ a\bullet & = & (f.a)\bullet \\
(12) & a = b & \equiv & a\bullet = b\bullet \\
(13) & (f \oplus^{\wedge} g).x & = & f.x \oplus g.x \\
(14) & (f \oplus^{\wedge} g) \circ h & = & (f \circ h) \oplus^{\wedge} (g \circ h) \\
(15) & 1_{\oplus^{\wedge}} & = & (1_{\oplus})\bullet
\end{array}$$

Conditionals

$$\begin{array}{lrcl}
(16) & x \langle\!\langle \mathbf{T} \rangle\!\rangle y & = & x \\
(17) & x \langle\!\langle \mathbf{F} \rangle\!\rangle y & = & y \\
(18) & (f \langle p \rangle g).x & = & f.x \langle\!\langle p.x \rangle\!\rangle g.x
\end{array}$$

$$
\begin{array}{lrcl}
(19) & \mathbf{true} & = & \mathbf{T}\bullet \\
(20) & \mathbf{false} & = & \mathbf{F}\bullet \\
(21) & \vee & = & \mathbf{or}^{\wedge} \\
(22) & \wedge & = & \mathbf{and}^{\wedge} \\
(23) & \neg & = & \mathbf{not}\circ \\
(24) & x \langle\!\langle b \rangle\!\rangle x & = & x \\
(25) & f \langle p \rangle f & = & f \\
(26) & f \langle \mathbf{true} \rangle g & = & f \\
(27) & f \langle \mathbf{false} \rangle g & = & g \\
(28) & f \langle \neg.p \rangle g & = & g \langle p \rangle f \\
(29) & f \langle p \wedge q \rangle g & = & (f \langle q \rangle g) \langle p \rangle g \\
(30) & f \langle p \vee q \rangle g & = & f \langle p \rangle (f \langle q \rangle g) \\
(31) & h \circ (f \langle p \rangle g) & = & h \circ f \langle p \rangle h \circ g \\
(32) & (f \langle p \rangle g) \circ h & = & f \circ h \langle p \circ h \rangle g \circ h
\end{array}
$$

Map

(33) $$(f \circ g)* = f* \circ g*$$

(34) Type hierarchy theorem

(35) For $h \in A \longrightarrow B$

$$
\begin{array}{rcl}
+\!\!+/ \circ h* & \in & \mathit{Tree}.A \longrightarrow \mathit{Tree}.B \\
+\!\!+/ \circ h* & \in & \mathit{List}.A \longrightarrow \mathit{List}.B \\
+\!\!+/ \circ h* & \in & \mathit{Bag}.A \longrightarrow \mathit{Bag}.B \\
+\!\!+/ \circ h* & \in & \mathit{Set}.A \longrightarrow \mathit{Set}.B
\end{array}
$$

(36) Type hierarchy theorem applied to $\oplus/$

Tree Identity Theorem

$$
\begin{array}{llcl}
(37) & +\!\!+/ \circ \tau* = 1_\circ & \in & [\![\mathit{Tree}.\alpha \longrightarrow \mathit{Tree}.\alpha]\!] \\
(38) & +\!\!+/ \circ \tau* = 1_\circ & \in & [\![\mathit{List}.\alpha \longrightarrow \mathit{List}.\alpha]\!] \\
 & +\!\!+/ \circ \tau* = 1_\circ & \in & [\![\mathit{Bag}.\alpha \longrightarrow \mathit{Bag}.\alpha]\!] \\
 & +\!\!+/ \circ \tau* = 1_\circ & \in & [\![\mathit{Set}.\alpha \longrightarrow \mathit{Set}.\alpha]\!]
\end{array}
$$

(39) $$f* = +\!\!+/ \circ (\tau \circ f)*$$

(40) $$\begin{aligned} * &\in [\![(\alpha \longrightarrow \beta) \longrightarrow List.\alpha \longrightarrow List.\beta]\!] \\ * &\in [\![(\alpha \longrightarrow \beta) \longrightarrow Bag.\alpha \longrightarrow Bag.\beta]\!] \\ * &\in [\![(\alpha \longrightarrow \beta) \longrightarrow Set.\alpha \longrightarrow Set.\beta]\!] \end{aligned}$$

<u>Filter</u>

(41) $$p\triangleleft = +\!\!+/ \circ (\tau \langle p \rangle 1_{+\!\!+^{\wedge}})*$$

(42) $$\mathbf{true}\triangleleft = 1_{\circ}$$

(43) Filter functionality

<u>Empty Rules</u>

(44) $$f* \circ 1_{+\!\!+^{\wedge}} = 1_{+\!\!+^{\wedge}}$$

(45) $$\oplus/ \circ 1_{+\!\!+^{\wedge}} = 1_{\oplus^{\wedge}}$$

(46) $$p\triangleleft \circ 1_{+\!\!+^{\wedge}} = 1_{+\!\!+^{\wedge}}$$

<u>One-point Rules</u>

(47) $$f* \circ \tau = \tau \circ f$$

(48) $$\oplus/ \circ \tau = 1_{\circ}$$

(49) $$p\triangleleft \circ \tau = \tau \langle p \rangle 1_{+\!\!+^{\wedge}}$$

<u>Join Rules</u>

(50) $$f*.(s +\!\!+ t) = (f*.s) +\!\!+ (f*.t)$$

(51) $$\oplus/.(s +\!\!+ t) = (\oplus/.s) \oplus (\oplus/.t)$$

(52) $$p\triangleleft.(s +\!\!+ t) = (p\triangleleft.s) +\!\!+ (p\triangleleft.t)$$

(53) $$x \oplus y = \oplus/.(\tau.x +\!\!+ \tau.y)$$

(54) Type of quantifiers

<u>Promotability</u>

(55) $\oplus \rightarrow \otimes$-distributive

(56) $\oplus \rightarrow \otimes$-promotable

(57) Promotability theorem

(66) $g \circ f$ promotability

(67) If $f$ is $\oplus\!\rightarrow\!\otimes$-promotable then

$$f \circ \oplus/ \circ t* \;=\; \otimes/ \circ (f \circ t)*$$

(68) $$\oplus/ \circ +\!\!+/ \;=\; \oplus/ \circ \oplus/*$$

(69) $$f* \circ +\!\!+/ \;=\; +\!\!+/ \circ f**$$

(70) $$p\triangleleft \circ +\!\!+/ \;=\; +\!\!+/ \circ p\triangleleft*$$

(71) $$f* \circ p\triangleleft \circ +\!\!+/ \;=\; +\!\!+/ \circ (f* \circ p\triangleleft)*$$

(72) $$\oplus/ \circ f* \circ +\!\!+/ \;=\; \oplus/ \circ (\oplus/ \circ f*)*$$

(73) $$\oplus/ \circ f* \circ p\triangleleft \circ +\!\!+/ \;=\; \oplus/ \circ (\oplus/ \circ f* \circ p\triangleleft)*$$

Bird's homomorphism lemma

(74) $$h \text{ is } +\!\!+\!\rightarrow\!\oplus\text{-promotable} \;\equiv\; h \;=\; \oplus/ \circ (h \circ \tau)*$$

(76) $$\oplus/ \;=\; \oplus/ \circ 1_\circ*$$

Range translation

(77) $$p\triangleleft \circ t* \;=\; t* \circ (p \circ t)\triangleleft$$

(78) $$\oplus/ \circ f* \circ p\triangleleft \circ t* \;=\; \oplus/ \circ (f \circ t)* \circ (p \circ t)\triangleleft$$

Trading rules

(80) $$\oplus/ \circ f* \circ p\triangleleft \;=\; \oplus/ \circ (f\langle p\rangle 1_{\oplus\hat{}})*$$

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

Identity rules

(87) $$1_{\oplus\hat{}} \;=\; \oplus/ \circ (1_{\oplus\hat{}})*$$

(88) $$\mathbf{false}\triangleleft \;=\; 1_{+\!\!+\hat{}}$$

(89) $$\oplus/ \circ f* \circ \mathbf{false}\triangleleft \;=\; 1_{\oplus\hat{}}$$

Lifting

(90) Lifting theorem

$$(93)\qquad (f{*}\circ)\ \circ\ (+\!\!+\hat{}/) \;=\; +\!\!+\hat{}/\ \circ\ (f{*}\circ{*})$$
$$(94)\qquad (p{\triangleleft}\circ)\ \circ\ (+\!\!+\hat{}/) \;=\; +\!\!+\hat{}/\ \circ\ (p{\triangleleft}\circ{*})$$
$$(95)\qquad (\oplus/\circ)\ \circ\ (+\!\!+\hat{}/) \;=\; \oplus\hat{}/\ \circ\ (\oplus/\circ{*})$$
$$(96)\qquad (h\tilde{\circ})\ \circ\ \oplus\hat{}/ \;=\; \oplus\hat{}/\ \circ\ ((h\tilde{\circ}){*})$$

Associativity and Symmetry

The properties below assume that $\oplus$ and $+\!\!+$ are associative and symmetric.

$$(97)\qquad \oplus/\ \circ\ (f \oplus\hat{}\ g){*} \;=\; (\oplus/\ \circ\ f{*}) \oplus\hat{}\ (\oplus/\ \circ\ g{*})$$
$$(98)\qquad \oplus/\ \circ\ (f \oplus\hat{}\ g){*}\ \circ\ p{\triangleleft} \;=\; (\oplus/\ \circ\ f{*}\ \circ\ p{\triangleleft}) \oplus\hat{}\ (\oplus/\ \circ\ g{*}\ \circ\ p{\triangleleft})$$

Range Splitting

$$(99)\qquad (p{\triangleleft}) +\!\!+\hat{}\ (\neg.p){\triangleleft} \;=\; 1_{\circ}$$
$$(100)\qquad \oplus/\ \circ\ f{*}\ \circ\ p{\triangleleft} \;=\; (\oplus/\ \circ\ f{*}\ \circ\ (p \wedge q){\triangleleft}) \oplus\hat{}\ (\oplus/\ \circ\ f{*}\ \circ\ (p \wedge \neg.q){\triangleleft})$$

Cartesian Product

$$(106)\qquad \oplus/\ \circ\ ((\oplus\hat{}/\ \circ\ h{*}\ \circ\ r{\triangleleft}).t){*}\ \circ\ p{\triangleleft} \;=\; (\oplus\hat{}/\ \circ\ g{*}\ \circ\ r{\triangleleft}).t$$
$$\text{where } g.x = \oplus/\ \circ\ (h.x){*}\ \circ\ p{\triangleleft}.$$

The properties below assume that $\oplus$ and $+\!\!+$ are idempotent (as well as being associative and symmetric).

Range Disjunction

$$(107)\qquad (p \vee q){\triangleleft} \;=\; p{\triangleleft} +\!\!+\hat{}\ q{\triangleleft}$$
$$(108)\qquad \oplus/\ \circ\ f{*}\ \circ\ (p \vee q){\triangleleft} \;=\; (\oplus/\ \circ\ f{*}\ \circ\ p{\triangleleft}) \oplus\hat{}\ (\oplus/\ \circ\ f{*}\ \circ\ q{\triangleleft})$$

Generalised Range Disjunction

$$(109)\qquad {\triangleleft}\ \circ\ \vee/ \;=\; +\!\!+\hat{}/\ \circ\ ({\triangleleft}{*})$$
$$(110)\qquad \oplus/\ \circ\ f{*}\ \circ\ ((\vee/\ \circ\ h{*}\ \circ\ r{\triangleleft}).t){\triangleleft} \;=\; (\oplus\hat{}/\ \circ\ g{*}\ \circ\ r{\triangleleft}).t$$
$$\text{where } g.x = \oplus/\ \circ\ f{*}\ \circ\ (h.x){\triangleleft}$$

# References


[1] R.C. Backhouse. *Program Construction and Verification.* Prentice-Hall International, London, 1986.

[2] R.C. Backhouse and B.A. Carré. Regular algebra applied to path-finding problems. *Journal of the Institute of Mathematics and its Applications*, 15:161–186, 1975.

[3] R.C. Backhouse, P. Chisholm, and G. Malcolm. Do-it-yourself type theory, part 1. *EATCS Bulletin*, 34, February 1988.

[4] R.C. Backhouse, P. Chisholm, and G. Malcolm. Do-it-yourself type theory, part 2. *EATCS Bulletin*, 35, June 1988.

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

[6] 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.

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

[8] R.S. Bird. The promotion and accumulation strategies in transformational programming. *ACM. Transactions on Programming Languages and Systems*, 6(4):487–504, 1984.

[9] R.S. Bird. Transformational programming and the paragraph problem. *Science of Computing Programming*, 6:159–189, 1986.

[10] R.S. Bird, 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.

[11] 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.

[12] P. Chisholm. *A Theory of Finite Sets in Constructive Type Theory.* Technical Report, Department of Computer Science, Heriot-Watt University, 1987.

[13] E.W. Dijkstra. *The calculus of boolean structures (Part1).* Technical Report EWD 1002, Department of Computer Sciences,The University of Texas at Austin, March 1987.

[14] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[15] E.W. Dijkstra and W.H.J. Feijen. *Een Methode van Programmeren.* Academic Service, Den Haag, 1984. Now available as *A Method of Programming*, Addison-Wesley, Reading, Mass., 1988.

[16] D. Gries. *The Science of Programming.* Springer-Verlag, New York, 1981.

[17] K. Iverson. *A Programming Language.* John Wiley & Sons, New York, 1962.

[18] D.E. Knuth. *Surreal Numbers.* Addison-Wesley, Reading, Mass., 1974.

[19] P. Martin-Löf. Constructive mathematics and computer programming. In C.A.R. Hoare and J.C. Shepherdson, editors, *Mathematical Logic and Computer Programming*, pages 167–184, Prentice-Hall, 1984.

[20] 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.

[21] R. Milner. A theory of type polymorphism in programming. *J. Comp. Syst. Scs.*, 17:348–375, 1977.

[22] A.J.M. van Gasteren and E.W. Dijkstra. *On notation.* Technical Report AvG65a/EWD950a, Department of Mathematics and Computing Science, Eindhoven University of Technology, January 1986.