1991

M.M. Fokkinga

Datatype laws without signatures

Computer Science/Department of Algorithmics and Architecture Report CS-R9133 June

CWI, nationaal instituut voor onderzoek op het gebied van wiskunde en informatica

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

Datatype Laws without Signatures

Maarten M. Fokkinga*

Using the categorical notion of 'functor' one may define the notion of data type (algebra) without being forced to introduce a signature, i.e., names for the individual sorts (types) and operations involved. This has proved to be advantageous for those theory developments where one is not interested in the syntactic appearance of an algebra. We present a categorical formalisation of the notion of 'law' for data types, that has the same property: no signature is needed. Thus we are able to prove some general theorems in quite a simple way.

1991 CR Classification System: D.1.1, D.3.3, F.3.2-3.
1980 Mathematics Subject Classification: 18A05, 69D11, 69D33, 69F32, 69F33.
Key Words and Phrases: datatype, algebra, signature, equational specification, category, functor.

* Research done while at the CWI, Amsterdam.
Current address: Twente University, dept. INF, P.O. Box 217, 7500 AE Enschede, The Netherlands; e-mail: fokkinga@cs.utwente.nl

Report CS-R9133
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1 Introduction

Most mathematical formalisations of the intuitive notion of 'data type' define that notion as a (many-sorted) algebra, possibly provided with some (conditional) equations, which we call 'laws'. Such algebras themselves are often formalised with help of the notion of 'signature' or, more categorically and slightly more abstract, 'sketch'. A signature gives the syntactic appearance of the algebra; it gives the names of the sorts (types), the names of the operations and constants, and for each operation a syntactic indication of its argument and result types. No doubt, signatures are indispensable for large scale programming tasks, and a theory that deals with signatures may be quite useful. Such a theory contains theorems on aspects of name-clashes, renaming, scope rules, persistency and so on. However, sometimes we would like to be able to abstract from syntactic aspects, e.g., when investigating the existence of certain kinds of algebras, or the (semantic) relations between algebras. In fact, one should abstract from naming even in the definition of such basic concepts as 'homomorphism'. For the lawless case this is possible indeed, thanks to the notion of *functor*. A functor allows you to characterise the type structure of an algebra without being forced to name the sort or any of the operations involved. Functors satisfy just one or two very simple axioms, and —almost unbelievable— that is all that is needed to develop a large body of useful theorems about algebras. The problem for which we propose a solution, is the following.

> Formalise the notion of 'law' (an equation or conditional equation that should be satisfied by the operations of the algebra) without introducing signatures, in particular naming and setting up a syntax of terms.

Remarkably, in all texts where functors are used to characterise algebras, signatures (or sketches) are introduced when it comes to laws. Clearly, this is a hindrance to theory development, since it forces to deal with aspects (syntax) that should have been abstracted from. About the use of functors to describe algebras Pierce [16, remark 1.11.14] explicitly says:

> The framework has apparently never been extended to include algebras with equations.

We shall present a definition of 'law' that avoids naming, and is of the same simplicity as the definition of 'functor'. To be specific, a law shall be just a mapping of morphisms (called '*transformer*' in analogy with 'functor') that satisfies a particular so-called *promotion* property. Using this definition we shall prove several theorems that everybody expects to be true. In fact, the simplicity of the proofs of these theorems demonstrates the success of our formalisation.

The remainder of the paper is organised as follows. We start by recalling the categorical, signatureless formalisation of 'algebra' in Section 2. Then we give the definition of 'transformer' and 'law' in Section 3, and provide some rationale for it. In the section that follows we investigate the expressiveness of transformers and laws; we also show that

the crucial promotion property of transformers is related to naturality and Wadler's Theorems For Free theorem. Then there come a series of sections in which we demonstrate the use of the promotion property of transformers: Section 5 for one half of a Birkhoff characterisation of the classes that can be specified with laws, Section 6 in which we use theorems from the literature to show the existence of initial lawful algebras, Section 7 in which it is shown that any colimit is in fact a lawful algebra, and finally Section 8 where we briefly discuss equational specifications of data types. In order to be truly general we work in a categorical setting; we assume that the reader is familiar with the basic notions and concepts of category theory.

2 Preliminaries

Apart from giving our notational conventions, we give a brief description of the formalisation of lawless algebras without signatures or sketches. A thorough discussion of this formalisation is given by Lehmann and Smith [10], Malcolm [11, 12], and Fokkinga and Meijer [6]. Also Manes and Arbib [13] and Pierce [16] give a brief description.

Nomenclature Variables $\mathbf{A}, \mathbf{B}, \mathbf{C}$ vary over categories; $a, b, c, \ldots$ vary over objects; $f, g, h, j, \ldots, p, q, \ldots, \alpha, \beta, \phi, \psi, \ldots$ vary over morphisms (actually, $\phi, \psi, \ldots$ vary over algebras and α, β over initial algebras); $F, G, H, J, K, \ldots$ vary over functors, and μF shall denote a distinguished initial F-algebra, assuming it exists. We use x, y, z to denote various entities. Formula $x : a \to_{\mathbf{A}} b$ means that x is a morphism in $\mathbf{A}$ with source a and target b ($\mathrm{src}_{\mathbf{A}} x = a$ and $\mathrm{tgt}_{\mathbf{A}} x = b$). If $\dagger$ is a bifunctor, then $F \dagger G$ denotes the monofunctor given by $(F \dagger G)x = Fx \dagger Gx$; this will occur often with $+$ and $\times$ for $\dagger$. The identity functor is denoted I; functor $I \times I$ is abbreviated to $\mathbb{I}$. A constant function (and functor) mapping each argument onto x (or the identity on x) is denoted $\underline{x}$.

We let $\mathbf{C}$ usually denote the base category, that is to say, we shall mostly suppress mentioning $\mathbf{C}$, certainly where it occurs as a subscript. In particular, $\to_{\mathbf{C}}$ is abbreviated to just $\to$. Unless stated explicitly otherwise, a morphism is a morphism in $\mathbf{C}$, and similarly for objects and endofunctors. We denote composition of the base category (and all the categories that inherit the composition of the base category) in diagramatic order: if $x : a \to b$ and $y : b \to c$ then $x ; y : a \to c$.

A typing of an expression is the indication of the source and target for each morphism and functor in the expression, and the indication of the category for each entity, such that the expression makes sense (e.g., targets and sources match in a composition). There exists an algorithm that derives for an expression the weakest possible typing requirements. Whenever we write an expression we tacitly assume that the (weakest) typing requirements are met.

Formula $\eta : F \dot{\to} G$ means that η is a natural transformation from F to G.

Product and coproduct In the examples and at several other places we assume that the base category $\mathbf{C}$ has products and coproducts. For products (in **Set**: cartesian

products) we use the notations $\times$, π_i, and $\vartriangle$ (and π_i^n in the examples). Here π_i is the projection onto the ith coordinate (numbering starts at 0), and $f \vartriangle g$ (f split g) is usually denoted $\langle f, g \rangle$; in **Set** we have $(f \vartriangle g \vartriangle h)x = (f\,x, g\,x, h\,x)$. For coproducts (in **Set**: disjoint unions) we use the notation $+$, inj_i, and $\triangledown$ (and inj_i^n in the examples). The usual notation for $f \triangledown g$ is $[f, g]$; in **Set** we have $(f \triangledown g \triangledown h)(inj_0\, x) = f\,x$, and so on. So, the typing reads

$$\begin{array}{lcll} f \vartriangle g & : & a \to b \times c & \text{whenever } f : a \to b \text{ and } g : a \to c \\ f \times g & : & a \times b \to c \times d & \text{whenever } f : a \to c \text{ and } g : b \to d \\ \pi_i & : & a_0 \times a_1 \to a_i & \\ f \triangledown g & : & a + b \to c & \text{whenever } f : a \to c \text{ and } g : b \to c \\ f + g & : & a + b \to c + d & \text{whenever } f : a \to c \text{ and } g : b \to d \\ inj_i & : & a_i \to a_0 + a_1\,. & \end{array}$$

A program like **if** p **then** f **else** g can be modeled by $p?\,; f \triangledown g$, where $p? : a \to a + a$ injects its argument into the left summand if p is true for it, and injects its argument into the right summand otherwise.

Algebra Let F be an endofunctor. An *F-algebra* is a morphism $\phi : Fa \to a$, for some object a called the carrier of ϕ.

Example (Trees) (In this and all the following examples we assume more or less that the base category is **Set**.) Let $*a$ be the set of trees over a, and let

$$\begin{array}{lcll} \Box & : & \mathbf{1} \to *a & \text{denote an atomic tree (“nil”)} \\ \tau & : & a \to *a & \text{denote the tip former} \\ +\!\!+ & : & \amalg *a \to *a & \text{denote the binary joining operation.} \end{array}$$

(A binary operation symbol like $+\!\!+$ will be parenthesised when it is a stand-alone argument for another binary operation like $;$ or $\triangledown$.) Take $\alpha = \Box \triangledown \tau \triangledown (+\!\!+)$ and $F_a = \underline{1} + \underline{a} + \amalg$, i.e., $F_a x = id_1 + id_a + x \times x$ for each function x. Then $\alpha : F_a *a \to *a$, so α is an F_a-algebra; the algebra of trees over a.

Another F_a-algebra is $\underline{0} \triangledown \underline{1} \triangledown (+)$; this one has carrier $\mathbb{N}$, the set of natural numbers. In the examples to come we let $e \triangledown f \triangledown (\oplus)$ be an arbitrary F_a-algebra with carrier b; so $e : \mathbf{1} \to b$, $f : a \to b$, $\oplus : \amalg b \to b$. (To be continued.) □

Given F-algebras ϕ and ψ, a morphism f is an *F-homomorphism from ϕ to ψ*, denoted $f : \phi \to_F \psi$, if $\phi\,; f = Ff\,; \psi$. Alg(F) is the category whose objects are F-algebras, whose morphisms are F-homomorphisms, and whose composition and identities are inherited from **C**; thus $\to_F$ is short for $\to_{\mathrm{Alg}(F)}$. An F-algebra α is called *initial* in Alg(F) if for each F-algebra ψ there exists precisely one homomorphism from α to ϕ; this one is denoted $(\!|\alpha := \phi|\!)_F$, or simply $(\!|\phi|\!)$ if F and α are understood, and it is called a *catamorphism*. We shall use α, β to denote initial algebras.

Example (Trees cont'd) Taking $\alpha = \Box \triangledown \tau \triangledown (+\!\!+)$ and $\phi = \underline{0} \triangledown \underline{1} \triangledown (+)$, the statement $size : \alpha \to_{F_a} \phi$ is equivalent to the three equations

$$\begin{array}{lcl} \Box\,;size & = & \underline{0} \\ \tau\,;size & = & \underline{1} \\ (+\!\!+)\,;size & = & \amalg size\,;(+)\,. \end{array}$$

In general such equations may have none, one, or more solutions for $size$. If α is initial, then these equations have precisely one solution: the function that yields the number of tips in the tree, and we can write $size = (\!|\Box \triangledown \tau \triangledown (+\!\!+) := \underline{0} \triangledown \underline{1} \triangledown (+)|\!)$ or simply $size = (\!|\underline{0} \triangledown \underline{1} \triangledown (+)|\!)$.

The three equations show that the effect of applying $size$ might be described as a systematic substitution $\Box, \tau, (+\!\!+) := \underline{0}, \underline{1}, (+)$, as suggested by the notation for catamorphisms. For example, $size((\tau x) +\!\!+ \Box) +\!\!+ (\tau y)) = ((1+0)+1)$. □

If an initial F-algebra exists we let μF denote one; for example we may have introduced the above algebra of trees by "let $\Box \triangledown \tau \triangledown (+\!\!+) = \mu F$".

Dually we might define a *G-co-algebra* to be a morphism $\phi : a \to Ga$, for some a called the *carrier of* ϕ. (The importance of co-algebras for programming is illustrated by Hagino [7], Malcolm [12], and Fokkinga and Meijer [6]. We shall hardly consider co-algebras in this paper.) There is also a generalisation that covers both algebras and co-algebras. A morphism $\phi : Fa \to Ga$, for some a, is called an *F,G-dialgebra*. For F,G-dialgebras ϕ and ψ, we say f is a dialgebra homomorphism from ϕ to ψ, denoted $f : \phi \to_{F,G} \psi$, if $\phi;Gf = Ff;\psi$. (Note the place of F and G in the latter formula: F describes the source structure of the two algebras, and G their target structure.) We shall use the notion of dialgebra mainly in the description of 'transformer' below.

Returning to algebras we list here some facts that follow from initiality and that we need in the sequel. Rule cata-CHARN (mnemonic for 'CHARACTERISATION') is just the *definition* of the initiality of α; the other rules follow quite easily. (In [5] we exploit such rules to prove various nontrivial results from category theory, by calculation rather than diagram chasing.) Suppose α is an initial F-algebra, and ϕ is just an F-algebra. Then, abbreviating $(\!|\alpha := \_|\!)$ to $(\!|\_|\!)$,

$$\begin{array}{lcll} x : \alpha \to_F \phi & \equiv & x = (\!|\phi|\!) & \text{cata-CHARN} \\ (\!|\phi|\!) : \alpha \to_F \phi & & & \text{cata-SELF} \\ (\!|\alpha|\!) = id & & & \text{cata-ID} \\ x, y : \alpha \to_F \phi & \Rightarrow & x = y & \text{cata-UNIQ} \\ x : \phi \to_F \psi & \Rightarrow & (\!|\phi|\!)\,;x = (\!|\psi|\!) & \text{cata-FUSION} \end{array}$$

(Remember that $x : \phi \to_F \psi$ just means that x satisfies the equation $\phi\,;x = Fx\,;\psi$.)

Any bifunctor $\dagger$ induces a monofunctor $a\dagger$ defined by $(a\dagger)x = id_a \dagger x$ for each morphism x. We define the *map functor induced by* $\dagger$, denoted $*$, by

$$\begin{array}{lcl} *a & = & \text{carrier of } \mu(a\dagger) \\ *f & = & (\!|f \dagger id_{*b}\,;\mu(b\dagger)|\!) \quad : \quad *a \to *b \quad \text{for } f : a \to b\,. \end{array}$$

One can prove that $*$ thus defined is a functor indeed.

Example (Trees cont'd) Take $x \dagger y = \underline{1} + \underline{x} + \amalg y$ so that the F_a defined earlier equals $a\dagger$, and let $\Box_a \triangledown \tau_a \triangledown (+\!\!+_a) = \mu(a\dagger)$. This is the algebra of trees over a, for arbitrary a. For $f : a \rightarrow b$ we have

$$\begin{array}{lcl} \Box_a \,;\, *f & = & \Box_b \\ \tau_a \,;\, *f & = & f \,;\, \tau_b \\ (+\!\!+_a) \,;\, *f & = & \amalg *f \,;\, (+\!\!+_b) \end{array}$$

so that $*f$ is the usual 'map f'. (The '$*$' in the previous examples denotes the map functor indeed.) We shall mostly omit the subscripts in $\Box_a, \tau_a, +\!\!+_a$. □

A lot more calculation properties of algebras and co-algebras, and their use in program development, have been given by Malcolm [11] and Fokkinga and Meijer [6]. The normal, co- and di- algebras defined above are 'single-sorted' with respect to the base category $\mathbf{C}$, since their carrier is just one object in $\mathbf{C}$. We won't discuss 'many-sortedness' in this paper (the algebras defined above are n-sorted in $\mathbf{A}$ if the base category $\mathbf{C}$ is an n-fold product category $\mathbf{A}^n$). The notion of datatype is discussed in a little more detail in Section 8.

3 Transformer and Law

Classically an equation for algebra ϕ is just a pair of terms built from variables, the constituent operations of ϕ, and some fixed standard operations. An equation is valid if the two terms are equal for all values of the variables. We are going to model a syntactic term as a morphism that has the values of the variables as source. For example, using the notation of the Tree example, the two terms 'x' and '$x +\!\!+ x$' (with variable x of type $*a$) are modeled by morphisms id and $id \vartriangle id \,;\, (+\!\!+)$ of type $*a \rightarrow *a$. So, an equation for ϕ is modeled by a pair of terms $(T\phi, T'\phi)$, T and T' being mappings of morphisms which we call *'transformer'* in analogy with 'functor'. This faces us with the following problem: what properties must we require of an arbitrary morphism mapping T in order that it models a classical syntactic term? Or rather, what properties of classical syntactic terms are semantically essential, and how can we formalise these as properties of a transformer T? Of course, T has to be well behaved with respect to typing (like functors). And besides that, the resulting morphism $T\phi$ should be built out of ϕ in a way that is independent of the properties of the particular ϕ itself and its carrier. For example, for I-algebras we disallow the following mappings as transformer.

$$\begin{array}{lcl} T\phi & = & \textbf{if } \phi \text{ has carrier } \mathbb{N} \textbf{ then } succ \textbf{ else } \phi \\ T\phi & = & \textbf{if } \phi \text{ is bijective } \textbf{then} \text{ the inverse of } \phi \textbf{ else } \phi . \end{array}$$

We disallow these not only for intuitive reasons, but also because with these mappings we cannot prove the things that we want to be true. The requirement that a transformer is a natural transformation (of some higher type) does meet our intuitive wish and enables us to prove the desirable theorems, but it would make various intuitive appealing laws

unexpressible (see Theorem (9)). So we need a weaker requirement to be imposed on a mapping in order that it can be said to model the intuitive notion of term. A way out is to introduce a syntax of terms, and require T to be expressed in that syntax. That is just what all authors have done up to now, and what we want to avoid.

Our solution is to impose a promotion property. This seems to be precisely what is needed to carry the proofs through. And it is also reasonable from an intuitive point of view. Let me try to explain it. (You may skip this informal explanation; the proofs of Theorem (5) and (6) are just the formalisation of the argument here.) Suppose $\phi : a \to a$ and $T\phi : Ha \to Ja$.

Following Meertens [15] we consider a term as a box with several input and output gates. Such boxes can be wired together to form composite boxes. You may imagine how the wiring for sequential composition (;) and parallel composition ($\times$) would look like. You can also easily construct several boxes for the duplication $\Delta = id \vartriangle id$, and for the swap $\bowtie = \pi_1 \vartriangle \pi_0$. Now imagine a box (term) $T\phi : Ha \to Ja$ built with several copies of a box for $\phi : a \to a$. Suppose you insert on each of the output lines a box for $f : a \to b$, thus forming a composite box $T\phi ; Jf : Ha \to Jb$. You can then shift each box for f along the wires in the direction of the input side, through all compositions, until it arrives just after a box for ϕ. If $\phi ; f = f ; \psi$ then you can replace the box for ϕ with one for ψ, and put the box for f just in front of ψ, and continue shifting the box for f along the lines. In this way, eventually, f is shifted ("promoted") to the input gates. Thus, if $\phi ; f = f ; \psi$ then you may expect that $T\phi ; Jf = Hf ; T\psi$. Generalising $\phi : a \to a$ and $\psi : b \to b$ to $\phi : Fa \to Ga$ and $\psi : Fb \to Gb$, you may expect in the same way that $\phi ; Gf = Ff ; \psi$ implies $T\phi ; Jf = Hf ; T\psi$. Using dialgebras we can formulate this as: $f : \phi \to_{F,G} \psi$ implies $f : T\phi \to_{H,J} T\psi$. We call this property 'promotion' since it generalises the (informal) notion of "promotion" used by Darlington [3] and Bird [2]. This leads us to the following definition.

Definitions (Transformer, Law)

(1) *A* transformer *of type* $(F \to G) \to (H \to J)$ *is a mapping* T *such that for all* a, T_a *maps each* $\phi : Fa \to Ga$ *onto a morphism* $T_a\phi : Ha \to Ja$, *and further*

$$\begin{array}{lll} \phi ; Gf = Ff ; \psi & \Rightarrow & T_a\phi ; Jf = Hf ; T_b\psi \\ \text{i.e.,} \quad f : \phi \to_{F,G} \psi & \Rightarrow & f : T_a\phi \to_{H,J} T_b\psi \end{array} \qquad \text{PROMOTION}$$

for all a, b, $f : a \to b$, $\phi : Fa \to Ga$, *and* $\psi : Fb \to Gb$. *We shall almost always omit the subscript in* T_a.

Rather than requiring that all entities above are in a base category $\mathbf{C}$, *it is only required that the expressions make sense. This leads to the following typing:* $\mathbf{A}, \mathbf{B}, \mathbf{C}$ *are categories,* $F, G : \mathbf{A} \to \mathbf{C}$, $H, J : \mathbf{B} \to \mathbf{C}$, a, b *in* $\mathbf{A}$, $f : a \to_{\mathbf{A}} b$, $\phi : Fa \to_{\mathbf{B}} Ga$, *and* $\psi : Hb \to_{\mathbf{C}} Jb$ [1].

[1] Categorically one can also say that a a transformer is just a functor $T : \mathrm{Alg}(F, G) \to \mathrm{Alg}(H, J)$ such

(2) *A* law *is a pair of transformers of the same type, called the* type *of the law. For a law* $E = (T, T')$ *we say* E holds for ϕ *if* $T\phi = T'\phi$. *Alternatively we also say* $E\phi$ holds *or* ϕ satisfies E; *a more formal notation would be* $\models E\phi$ *or* $\phi \models E$.

(3) *A* conditional law *is a pair* E, E' *of laws both having the same argument type* $F \to G$. *Such a law holds for* ϕ *if* $E\phi$ *implies* $E'\phi$. *(We shall hardly discuss conditional laws.)*

Mostly we will take $\mathbf{A} = \mathbf{B} = \mathbf{C}$ in applications of transformers, so that F, G, H, J are endofunctors on the base category $\mathbf{C}$. It is straightforward to extend the definitions in such a way that transformers and laws accept several arguments rather than one. Actually, this is already covered by the above definition: take $\mathbf{B}$ to be a suitable product category, $\mathbf{B} = \mathbf{B}_0 \times \mathbf{B}_1$. This will occur in Section 8.

Usage of laws If a law is prescribed for an F-algebra ϕ, then of course the law must have argument type $(F \to I)$, i.e., $G = I$. You may think that the definition of law and transformer is unnecessarily general for this application. However, in composing transformers you need the more general form with $G \neq I$, even though the entire composite transformer has $G = I$; see Theorem (6). A dual remark holds for co-algebras. As regards to the result type we can make a similar observation; in this case either $H = I$ or $J = I$ depending on the use of the law. We illustrate both possibilities for the use of a law in the following example; yet another use, related to the first one, is discussed in Section 8.

Example (Trees cont'd) Consider the law "$x + y = y + x$" which we shall formalise later. Here the result type of the two terms, viewed as functions of x and y, is $\mathbb{1} * a \to * a$ where $* a$ is the carrier. In this case $J = I$ (and $H = \mathbb{1}$) and the law induces a equivalence relation on $* a$ that is a congruence for the algebra, namely the least equivalence relation that contains all pairs $(x + y, y + x)$ (as indicated by the law) and is closed under the operations of the algebra (with (x, x') and (y, y') it also contains $(x + y, x' + y')$). Imposing the law on the algebra means to identify equivalent elements and to consider the induced quotient algebra.

Now consider law "$size\, x \bmod 2 = 0$" (also formalised later). Here the result type of the two terms, viewed as functions of x, is $* a \to \mathbb{N}$, so in this case $H = I$ (and $J = \underline{\mathbb{N}}$). Imposing the law on the algebra means to leave out from the carrier the trees with odd size and to look for an "induced subalgebra" (which might not exist at all). □

In the sequel we shall illustrate our notion of transformer and law mainly for the case $G = I = J$: applicable to algebras and meant to identify elements of the carrier. We are in fact primarily interested in the rôle of the PROMOTION rule for transformers, since we conjecture this to be the heart of the formalisation of the semantics of terms. Further applications of transformers and laws await future research.

---

that $UT = U$, where $U : \mathrm{Alg}(F, G) \to \mathbf{A}$ is the forgetful functor with $U\phi = a$ and $Uf = f$ (using the variables and typing of the definition).

Example (Trees cont'd) By making $\Box$ neutral for $+\!\!+$ (that is, making $\Box$ the identity for $+\!\!+$) it behaves properly as an 'empty' tree. By further imposing associativity of $+\!\!+$ the trees become effectively *lists*, or *sequences*. *Bags* result from imposing commutativity of $+\!\!+$, and, finally, *sets* are obtained if $+\!\!+$ is made absorptive (idempotent) as well. (Meertens [14] attributes this observation to H.J. Boom, and Backhouse [1] calls these types the Boom-hierarchy.) We shall show how the laws can be expressed as pairs of transformers. The laws are applicable to every $a\dagger$-algebra, not only to the initial one (the trees). Also the law for 'even size' of trees is formalised; this one has a feature not present in the others.

Let $\phi = e \triangledown f \triangledown (\oplus) : \mathbf{1} + a + \amalg *a \to *a$ be an arbitrary $a\dagger$-algebra. Observe that the constituent operations of ϕ can be expressed as follows.

$$\begin{array}{lcll} e & = & inj_0^3 ; \phi & : \ \mathbf{1} \to *a \\ f & = & inj_1^3 ; \phi & : \ a \to *a \\ \oplus & = & inj_2^3 ; \phi & : \ \amalg *a \to *a \, . \end{array}$$

So if we say $T(\phi) = \ldots e \ldots \oplus \ldots$ we actually mean the right hand side that is obtained by substituting the above definitions for $e, f,$ and $\oplus$.

Absorptivity In order to express $x \oplus x = x$ for all x in $*a$, take $T\phi = \Delta ; (\oplus)$ where $\Delta = id \vartriangle id$, and $T'\phi = id$. It is quite simple to verify rule PROMOTION.

Commutativity To express $x \oplus y = y \oplus x$ for all x, y in $*a$, we take $T\phi = \oplus$ and $T'\phi = \bowtie ; (\oplus)$, where the swap is defined by $\bowtie = \pi_1^2 \vartriangle \pi_0^2$. Again, PROMOTION is quite easily verified.

Neutrality To express $e \oplus x = x$ for all x in $*a$ we take $T\phi = (! ; e) \vartriangle id$ and $T'\phi = id$; here $! : *a \to \mathbf{1}$ is the unique morphism into the unit type (final object) $\mathbf{1}$. It's time to remark that verification of rule PROMOTION is hardly necessary: Theorems (6, 7, 8) allow you to conclude the validity of the rule by considering the way T and T' are composed out of basic transformers.

Associativity To express $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ for all x, y, z in $*a$, we take $T\phi = (\oplus) \times id ; (\oplus)$ and $T'\phi = rearrange ; id \times (\oplus) ; (\oplus)$, where $rearrange : (\alpha \times \beta) \times \gamma \to \alpha \times (\beta \times \gamma)$ is easily defined using projections and splits (left to the reader).

Even size A problem in expressing $(size\, x) \bmod 2 = 0$ is that *size* is not an operation of the algebra. Given that ϕ is the initial algebra we can take $(\!|\phi := \underline{0} \triangledown \underline{1} \triangledown (+)|\!)$ for *size*, as we have shown earlier. However, T should be applicable to any algebra ϕ, not just an initial one. In fact, it is a problem what "*size*" means at all if ϕ is not initial. One way out is this. First extend the algebra with an additional operation ψ specified by the 'defining equations' for *size*: "$\phi ; \psi = F\psi ; \underline{0} \triangledown \underline{1} \triangledown (+)$". This is discussed in detail in Section 8 and gives an F, G-bi-algebra (ϕ, ψ) for some G. Then we form, for any F, G-bi-algebra (ϕ, ψ), the law E suggested by

$$E(\phi, \psi) \quad = \quad \text{“ } \phi ; \psi ; \mathrm{mod}2 = \underline{0} \text{ ”} \, .$$

By a suitable instantiation of $\mathbf{A}, \mathbf{B}$ and the functors in the definition of transformer, we have that transformers are applicable to bi-algebras. $\Box$

After all these examples one might wonder whether there are morphism mappings that have type $(F \to G) \to (H \to J)$ for some F, G, H, J and are not transformers.

(4) Fact *The morphism mappings given at the beginning of the section are not transformers; the* PROMOTION *rule is not valid for them.*

4 Expressiveness of transformers and laws

We show that the PROMOTION rule for a transformer follows from the Theorems For Free theorem (provided it is applicable to the transformer). Further, the usual syntactic ways to compose terms are also applicable to transformers. Thus transformers are at least as expressible as syntactic terms. Also natural transformations of a higher type are transformers, but not conversely. Finally we show that laws are closed under conjunction.

* * *

(5) Theorem *Let T be any morphism mapping of type $\forall\alpha :: (F\alpha \to G\alpha) \to (H\alpha \to J\alpha)$, and suppose that the Theorems For Free theorem of Wadler [17] is applicable to T. Then T is a transformer.*

Proof We use the notation of Wadler [17] except for our choice of identifiers and the order of composition:

Any function f denotes a relation, defined by $(x,y) \in f \equiv f(x)=y$. Composition ; is extended to relations: $(x,z) \in R\,;S$ iff there exists an y for which $(x,y) \in R$ and $(y,z) \in S$. For relations R and S and relation mapping F, expressions $R \to S$ and $\forall r :: Fr$ denote a relation too:

$$\begin{array}{lcl} (f,g) \in (R \to S) & \equiv & R\,;g \subseteq f\,;S \\ & \equiv & \forall x,y :: \ (x,y) \in R \ \Rightarrow \ (f\,x, g\,y) \in S \\ (T,T') \in (\forall r :: Fr) & \equiv & \forall a,b,\ R{:}a \Leftrightarrow b :: \ (T_a, T'_b) \in FR. \end{array}$$

All morphisms (functions) are required to be total, so that $f \subseteq g$ equivales $f = g$.

The task is to prove that PROMOTION is valid for T. For this we argue

$$\begin{array}{ll} & T : \forall\alpha :: (F\alpha \to G\alpha) \to (H\alpha \to J\alpha) \\ \Rightarrow & \quad \text{Theorems for Free — applicability assumed} \\ & (T,T) \in \forall r :: (Fr \to Gr) \to (Hr \to Jr) \\ \equiv & \quad \text{definition } \forall \\ & \forall \text{reln } R : a \Leftrightarrow b. \ (T_a, T_b) \in (FR \to GR) \to (HR \to JR) \\ \equiv & \quad \text{definition } \to \text{ (second alternative)} \\ & \forall \text{reln } R : a \Leftrightarrow b. \ \forall\phi,\psi. \ (\phi,\psi) \in (FR \to GR) \ \Rightarrow \ (T_a\phi, T_b\psi) \in (HR \to JR) \\ \equiv & \quad \text{definition } \to \text{ (first alternative) at both sides} \end{array}$$

$$\forall \text{reln } R : a \Leftrightarrow b.\ \forall \phi, \psi.\ \ FR\,;\psi \subseteq \phi\,;GR \ \Rightarrow\ HR\,;T_b\psi \subseteq T_a\phi\,;JR$$

$\Rightarrow$ taking $R : a \Leftrightarrow b$ to be a function $f : a \to b$

$$\forall \text{fctn } f : a \to b.\ \forall \phi, \psi.\ \ Ff\,;\psi = \phi\,;Gf \ \Rightarrow\ Hf\,;T_b\psi = T_a\phi\,;Jf$$

which is exactly the required PROMOTION rule. □

One condition on T for the applicability of the Theorems For Free theorem is that T is lambda-definable (there may be also conditions on the category — this is not clear to me). We allow for any morphism mapping in the definition of transformer, even those that are not lambda-definable. Theorems For Free suggests that PROMOTION is a crucial property (and provides an alternative rationale for requiring this rule to hold for transformers). Moreover, working in a 'functional' categorical setting, it seems that Theorems For Free suggests no stronger property for transformers.

Composite transformers Here come some theorems showing how transformers may be composed to form transformers again.

(6) Theorem *The following equations on the left define transformers* I, $\underline{f}$, $\underline{\eta}$, $R;S$, $T \circ U$, *and* $(\!|\_|\!)$ *of type* $(F \to G) \to (H \to J)$, *provided that* R, S, T, U, V *are transformers and the well-formedness conditions on the right hold.*

$$\begin{array}{lcll}
I\phi & = & \phi & \left\{ H = F \text{ and } J = G \right. \\
\underline{f}\phi & = & f & \left\{ H = \underline{\mathrm{src}\, f} \text{ and } J = \underline{\mathrm{tgt}\, f} \right. \\
\underline{\eta}\phi & = & \eta & \left\{ \eta : H \dot{\to} J \right. \\
(R;S)\phi & = & R\phi\,;S\phi & \left\{ \begin{array}{l} R : (F \to G) \to (H \to K) \\ S : (F \to G) \to (K \to J) \end{array} \right. \text{for some } K \\
(T \circ U)\phi & = & T(U\phi) & \left\{ \begin{array}{l} T : (K \to L) \to (H \to J) \\ U : (F \to G) \to (K \to L) \end{array} \right. \text{for some } KL \\
(\!|_|\!)\,\phi & = & (\!|\mu F := \phi|\!) & \left\{ \begin{array}{l} G = I = J \text{ and } H = \underline{\mathrm{tgt}\ \mu F} \\ \mu F \text{ exists} \end{array} \right.
\end{array}$$

Proof The correct typing is immediate for all these transformers. As regards to the PROMOTION property of the composite transformer $T \circ U$ we argue:

$$T(U\phi)\,;Jf = Hf\,;T(U\psi)$$

$\Leftarrow$ PROMOTION for T

$$U\phi\,;Lf = Kf\,;U\psi$$

$\Leftarrow$ PROMOTION for U

$$\phi\,;Gf = Ff\,;\psi\,.$$

(You may wish to rephrase this calculation with the $\rightarrow_{F,G}$ notation.) As regards to PROMOTION for $(\!|\_|\!)$ we argue as follows. Let $a = \mathrm{tgt}\ \mu F$. Then

$$
\begin{array}{ll}
 & \underline{a} f \,;\, (\!|\_|\!)\phi = (\!|\_|\!)\psi \,;\, I f \\
\equiv & \quad \text{definition } (\!|\_|\!),\ \underline{a}, \text{ and } I \\
 & (\!|\mu F := \phi|\!) = (\!|\mu F := \psi|\!) \,;\, f \\
\Leftarrow & \quad \text{cata-FUSION} \\
 & \psi \,;\, f = F f \,;\, \phi \,.
\end{array}
$$

The other parts are proved similarly to $T \circ U$ above. Actually, that $\underline{f}$ is a transformer follows also from the fact that $\underline{\eta}$ is a transformer, since for any $f : a \rightarrow b$ we have $f : \underline{a} \rightarrow\!\!\!\!\cdot\ \underline{b}$. □

(7) Theorem *Let $F, G : \mathbf{A} \rightarrow \mathbf{B}$, $H, J : \mathbf{A} \rightarrow \mathbf{C}$, and $K : \mathbf{C} \rightarrow \mathbf{C}$ be functors. Let T be a transformer of type $(F \rightarrow G) \rightarrow (H \rightarrow J)$. Then T is also a transformer of type $(FK \rightarrow GK) \rightarrow (HK \rightarrow JK)$.*

Proof The typing requirement is easily verified. As regards to the PROMOTION property we argue:

$$
\begin{array}{ll}
 & f : T\phi \rightarrow_{HK,JK} T\psi \\
\equiv & \quad \text{unfold, fold} \\
 & Kf : T\phi \rightarrow_{H,J} T\psi \\
\Leftarrow & \quad \text{PROMOTION for transformer } T \text{ of type } (F \rightarrow G) \rightarrow (H \rightarrow J) \\
 & Kf : \phi \rightarrow_{F,G} \psi \\
\equiv & \quad \text{unfold, fold} \\
 & f : \phi \rightarrow_{FK,GK} \psi
\end{array}
$$

as required. □

The next theorem shows that each functor is a transformer. We use π_0 and π_1 as projection functors from $\mathbf{A} \times \mathbf{A}$ to $\mathbf{A}$, for any category $\mathbf{A}$. For a functor K we define the family of morphism mappings $K_{a,b}$ by $K_{a,b}\phi = K\phi$ whenever $\phi : a \rightarrow b$. The subscript in $K_{a,b}$ is completely superfluous since it is fully determined by the argument: $a = \mathrm{src}\phi$ and $b = \mathrm{tgt}\phi$. Therefore we make no syntactic distinction between K and the family of mappings $K_{a,b}$; the latter is also denoted K.

(8) Theorem *Let $K : \mathbf{B} \rightarrow \mathbf{C}$ be a functor. Let $\mathbf{A} = \mathbf{B} \times \mathbf{B}$, and $F, G = \pi_0,\ \pi_1$ (functors of type $\mathbf{A} \rightarrow \mathbf{B}$) and $H, J = K \circ \pi_0,\ K \circ \pi_1$ (functors of type $\mathbf{A} \rightarrow \mathbf{C}$). Then K is a transformer of type $(F \rightarrow G) \rightarrow (H \rightarrow J)$.*

Proof The typing requirement for K is met:

$$
\begin{array}{ll}
& \forall a \text{ in } \mathbf{A},\ \phi : Fa \rightarrow_{\mathbf{B}} Ga :: \quad K_{Fa,Ga}\phi : Ha \rightarrow_{\mathbf{C}} Ja \\
\equiv & \quad \text{unfold } \mathbf{A} \text{ and } F, G, H, J \text{ and } K_{-,-} \\
& \forall b, c \text{ in } \mathbf{B},\ \phi : b \rightarrow_{\mathbf{B}} c :: \quad K\phi : Kb \rightarrow_{\mathbf{C}} Kc \\
\equiv & \quad \text{functoriality of } K \\
& \mathsf{true}\,.
\end{array}
$$

To check the PROMOTION property, let

$$a, b \text{ in } \mathbf{A}, \quad f : a \rightarrow_{\mathbf{A}} b, \quad \phi : Fa \rightarrow_{\mathbf{B}} Ga, \quad \psi : Hb \rightarrow_{\mathbf{B}} Jb$$

be arbitrary. We argue:

$$
\begin{array}{ll}
& f : K_a\phi \rightarrow_{H,J} K_b\psi \\
\equiv & \quad \text{observe that } H, J = KF, KG \text{ and } K_a, K_b = K, K \\
& f : K\phi \rightarrow_{KF,KG} K\psi \\
\Leftarrow & \quad \text{general theorem (for arbitrary } F, G) \\
& f : \phi \rightarrow_{F,G} \psi\,,
\end{array}
$$

as required. □

Although it was not needed in the proof, it may be instructive to work out the PROMOTION property for arbitrary functor $K : \mathbf{B} \rightarrow \mathbf{C}$ when considered as a transformer of type $(\pi_0 \rightarrow \pi_1) \rightarrow (K \circ \pi_0 \rightarrow K \circ \pi_1)$. To this end observe that here, apparently, $\pi_0, \pi_1 : \mathbf{B} \times \mathbf{B} \rightarrow \mathbf{B}$, and that any morphism f in $\mathbf{B} \times \mathbf{B}$ has the form (g, h) for some morphisms g, h in $\mathbf{B}$. Property PROMOTION thus reads

$$(g, h) : \phi \rightarrow_{\pi_0,\pi_1} \psi \quad \Rightarrow \quad (g, h) : K\phi \rightarrow_{K\circ\pi_0,\ K\circ\pi_1} \psi$$

i.e.,

$$\phi\,;h = g\,;\psi \quad \Rightarrow \quad K\phi\,;Kh = Kg\,;K\psi\,.$$

This is clearly valid for any functor K, as asserted by the theorem.

From all these theorems we conclude that for all conventional laws there is no need to check PROMOTION explicitly: the transformers of such laws are built entirely by the compositions of the theorems. In particular this holds for morphisms and natural transformations like projections, injections, split, junc, product and coproduct and so on.

Naturality Next we will show that natural transformations of type $(F \ddot{\rightarrow} G) \rightarrow (H \ddot{\rightarrow} J)$ are transformers. Let us first explain the two composite functors $(F \ddot{\rightarrow} G)$ and $(H \ddot{\rightarrow} J)$. The infix written bifunctor $\_\rightarrow\_ : \mathbf{C}^{op} \times \mathbf{C} \rightarrow \mathbf{Set}$ is defined as follows (it is the Hom-functor). For objects a, b and morphisms $f : a \rightarrow b$ and $g : c \rightarrow d$,

$$
\begin{array}{lcl}
(a \rightarrow b) & = & \{x |\ \ x : a \rightarrow_{\mathbf{C}} b\} \\
(f \rightarrow g) & = & \lambda(x ::\ \ f\,;x\,;g) \quad : \quad (b \rightarrow c) \rightarrow (a \rightarrow d)\,.
\end{array}
$$

Notice that $\rightarrow$ is contravariant in its first argument, and that $x : a \rightarrow b$ equivales $x \in (a \rightarrow b)$ (thus justifying our choice of notation). We also introduce the notational convention that for binary operation $\oplus$ we write $f \ddot{\oplus} g$ for the binary operation mapping x, y onto $fx \oplus gy$.[2] Now the functors in the (natural transformation) type of T are fully defined. Working out all levels in detail we find

$$
\begin{array}{ll}
 & T : (F \ddot{\rightarrow} G) \dot{\rightarrow} (H \ddot{\rightarrow} J) \\
\equiv & \quad \text{ntrf: for all } a, b, c, d \text{ and } f : a \rightarrow b \text{ and } g : c \rightarrow d \\
 & (Ff \rightarrow Gg) \,;\, T_{ad} = T_{bc} \,;\, (Hf \rightarrow Jg) \\
\equiv & \quad \text{extensionality: for all } \phi \in (b \rightarrow c) \\
 & ((Ff \rightarrow Gg) \,;\, T_{ad})\phi = (T_{bc} \,;\, (Hf \rightarrow Jg))\phi \\
\equiv & \quad \text{application, composition, hom-functor} \\
 & T_{ad}(Ff \,;\, \phi \,;\, Gg) = Hf \,;\, T_{bc}\phi \,;\, Jg \,.
\end{array}
$$

That is, natural transformation T satisfies a two-sided fusion law. (An adjunction between F and J is nothing but such a natural transformation (with $G = I = H$) that has an inverse.)

(9) Theorem *Any natural transformation $T : (F \ddot{\rightarrow} G) \dot{\rightarrow} (H \ddot{\rightarrow} J)$ is a transformer of type $(F \rightarrow G) \rightarrow (H \rightarrow J)$. The converse is not true.*

Proof Let T be a natural transformation as indicated in the theorem. Defining $T_a = T_{aa}$ we find for $\phi : Fa \rightarrow Ga$ that $T_a\phi : Ha \rightarrow Ja$, as required. It remains to verify the PROMOTION property.

$$
\begin{array}{ll}
 & T\phi \,;\, Jf = Hf \,;\, T\psi \\
\equiv & \quad \text{naturality } T, \text{ in lhs with } f, g := id, f \text{ and in rhs with } f, g := f, id \\
 & T(\phi \,;\, Gf) = T(Ff \,;\, \psi) \\
\Leftarrow & \quad \text{Leibniz} \\
 & \phi \,;\, Gf = Ff \,;\, \psi \,.
\end{array}
$$

To show that the converse is not true, consider any $\eta : H \dot{\rightarrow} J$. Then $\underline{\eta}$ is a transformer of type $(F \rightarrow G) \rightarrow (H \rightarrow J)$. It is not a natural transformation of type $(F \ddot{\rightarrow} G) \dot{\rightarrow} (H \ddot{\rightarrow} J)$ since the typing is not correct; this is also apparent from the two-sided fusion law that now simplifies to

$$\eta \quad = \quad Hf \,;\, \eta \,;\, Jg$$

which should hold for any a, b, c, d and $f : a \rightarrow b$ and $g : c \rightarrow d$ — clearly impossible in general. For a counterexample, take $\eta = id : I \dot{\rightarrow} I$. □

[2] More generally, for $\oplus : Fa \rightarrow b$ (where a may be a vector) one might define the lifted version $\dot{\oplus} : F(x \rightarrow a) \rightarrow x \rightarrow b$ (for an arbitrary non-vector x) which is always denoted just $\oplus$ in this paper, and the F-lifted version $\ddot{\oplus} : F(y \rightarrow a) \rightarrow Fy \rightarrow b$ (for an arbitrary vector y of the same length as a). These are discussed by Van der Woude [18]. For binary operations, $F = \mathbb{I}$ and each of $a, y, (y \rightarrow a)$ is a single object, or $F = \times$ and each of $a, y, (y \rightarrow a)$ is a pair of objects. For binary operations we also have $f \ddot{\oplus} g = f \times g \,;\, (\oplus)$.

Conjunction Next we show that there are two ways of representing the conjunction of several laws. The choice between these two ways is dictated by the desired result type of the law.

For mappings $T_i : (F \to G) \to (H_i \to J)$ $(i = 0, 1)$ we define $T_0 \triangledown T_1$ by

$$(T_0 \triangledown T_1)\phi \quad = \quad T_0\phi \triangledown T_1\phi \quad : \quad H_0 a + H_1 a \to Ja$$

for any a and $\phi : Fa \to Ga$. It follows that $T_0 \triangledown T_1$ is a morphism mapping of type $(F \to G) \to (H_0 + H_1 \to J)$. The composite $S_0 \triangle S_1$ is defined similarly, and we have

$$S_0 \triangle S_1 \quad : \quad (F \to G) \to (H \to J_0 \times J_1)$$

for $S_i : (F \to G) \to (H \to J_i)$. Of course we need to assume that the category has coproducts or products respectively.

(10) Theorem *Let $T_i, T_i' : (F \to G) \to (H_i \to J)$ be transformers for $i = 0, 1$. Then $T_0 \triangledown T_1$ is a transformer, and*

$$(T_0 \triangledown T_1)\phi = (T_0' \triangledown T_1')\phi \quad \equiv \quad T_0\phi = T_0'\phi \ \wedge \ T_1\phi = T_1'\phi .$$

Similarly, for $S_i, S_i' : (F \to G) \to (H \to J_i)$, mapping $S_0 \triangle S_1$ is a transformer, and

$$(S_0 \triangle S_1)\phi = (S_0' \triangle S_1')\phi \quad \equiv \quad S_0\phi = S_0'\phi \ \wedge \ S_1\phi = S_1'\phi .$$

Proof The equivalences follow from the properties for product and coproduct. For the PROMOTION property for $T_0 \triangledown T_1$ we argue as follows. Let $\phi ; Gf = Ff ; \psi$. Then

$$\begin{array}{ll} & (T_0 \triangledown T_1)\phi ; Jf \\ = & (T_0\phi ; Jf) \triangledown (T_1\phi ; Jf) \\ = & \quad \text{PROMOTION for } T_i \text{ at both sides} \\ & (H_0 f ; T_0\psi) \triangledown (H_1 f ; T_1\psi) \\ = & (H_0 + H_1)f ; (T_0 \triangledown T_1)\psi . \end{array}$$

The proof for $S_0 \triangle S_1$ is similar. □

Of course we also have arbitrary infinite conjunctions of laws, provided the category has arbitrary infinite coproducts or products. In case $H = J = I$ it depends on the use of the laws (more precisely the desired result type) whether we have to model conjunction by means of coproduct or product!

5 One half of a Birkhoff characterisation

Let F and H be endofunctors and $E = (T, T')$ be a law of type $(F \to I) \to (H \to I)$, fixed throughout this section. We define $\mathrm{Alg}(F, E)$ as the full subcategory of $\mathrm{Alg}(F)$ containing all and only those F-algebras for which law E is valid. A "Birkhoff characterisation" is a characterisation of the classes (subcategories) that can be specified by means of a single law. For example, a characterisation might be an equivalence like: for any class $\mathcal{A}$ of F-algebras,

$\mathcal{A} = \mathrm{Alg}(F, E)$ for some law E

if and only if

$\mathcal{A}$ is closed under subalgebras, homomorphisms, and products.

We shall give one half of such an equivalence (the easy half): some closure properties of any $\mathrm{Alg}(F, E)$. Some care is needed in defining subalgebras and homomorphic images since we wish to work in an arbitrary category, and not just in **Set** where all kinds of extra properties hold. The two notions that we define are dual to each other.

Subalgebra Given F-algebras ϕ and ψ, we say ϕ is a *subalgebra* of ψ if there exists an $f : \phi \to_F \psi$ which is monic in $\mathbf{C}$. A subcategory $\mathcal{A}$ of $\mathrm{Alg}(F)$ is *closed under subalgebras* if each subalgebra of an algebra in $\mathcal{A}$ is in $\mathcal{A}$ too. More in spirit with the position that in any category the morphisms are important and the objects play only an auxiliary rôle, we define also: $\mathcal{A}$ is *closed under incoming monos* if f is a morphism in $\mathcal{A}$ whenever $f : \phi \to_F \psi$ is monic in $\mathbf{C}$ and ψ is in $\mathcal{A}$.

(11) Theorem $\mathrm{Alg}(F, E)$ *is closed under subalgebras (i.e., under incoming monos).*

Proof Suppose $f : \phi \to_F \psi$ is monic, and ψ is in $\mathrm{Alg}(F, E)$. We show that $E\phi$ holds.

$$
\begin{array}{ll}
 & T\phi = T'\phi \\
\Leftarrow & \quad f \text{ monic} \\
 & T\phi \,;\, f = T'\phi \,;\, f \\
\equiv & \quad \textsc{Promotion} \text{ (condition '}\phi \,;\, f = Ff \,;\, \psi\text{' is satisfied) at both sides} \\
 & Hf \,;\, T\psi = Hf \,;\, T'\psi \\
\Leftarrow & \quad \text{Leibniz, and } \psi \text{ in } \mathrm{Alg}(F, E) \\
 & \mathsf{true}\,.
\end{array}
$$

So ϕ is in $\mathrm{Alg}(F, E)$ and, since $\mathrm{Alg}(F, E)$ is a full subcategory of $\mathrm{Alg}(F)$, f is a morphism in $\mathrm{Alg}(F, E)$ as well. □

Homomorphisms Consider $f : \phi \to_F \psi$. Working in **Set** we can define the homomorphic image of ϕ under f as the algebra ψ restricted to the range of f. In general we cannot do so, since categorically we have no points available. Working with *varieties*, as Lehmann [8, 9] does, the corresponding closure property says that with ϕ also ψ is in the class. That is certainly not true for $\mathrm{Alg}(F, E)$, as we shall argue after the theorem. Our way out is to consider epic homomorphisms. We define: A subcategory $\mathcal{A}$ of $\mathrm{Alg}(F)$ is *closed under homomorphic images* if ψ is in $\mathcal{A}$ whenever there exists an $f : \phi \to_F \psi$ which is epic in $\mathbf{C}$ and ϕ is in $\mathcal{A}$. And, $\mathcal{A}$ is *closed under outgoing epis* if f is a morphism in $\mathcal{A}$ whenever $f : \phi \to_F \psi$ is epic and ϕ is in $\mathcal{C}$.

(12) Theorem *Suppose H (from the type of E) preserves epis. Then $\mathrm{Alg}(F, E)$ is closed under homomorphic images (i.e., under outgoing epis).*

Proof Let $f : \phi \to_F \psi$ be epic, with ϕ in $\mathrm{Alg}(F, E)$.

$$
\begin{array}{ll}
 & T\psi = T'\psi \\
\Leftarrow & \quad Hf \text{ is epic} \\
 & Hf\,;T\psi = Hf\,;T'\psi \\
\equiv & \quad \textsc{Promotion} \text{ (condition '} f : \phi \to_F \psi \text{' is satisfied) at both sides} \\
 & T\phi\,;f = T'\phi\,;f \\
\equiv & \quad \text{assumption: } \phi \text{ in } \mathrm{Alg}(F, E) \\
 & \mathsf{true}.
\end{array}
$$

So ψ is in $\mathrm{Alg}(F, E)$, and since $\mathrm{Alg}(F, E)$ is a full subcategory of $\mathrm{Alg}(F)$, f is a morphism in $\mathrm{Alg}(F, E)$. □

For later use we mention the following result; its proof is part of the above one.

(13) Lemma *Let ϕ in $\mathrm{Alg}(F, E)$, and $f : \phi \to_F \psi$. Then $E\psi$ holds "on the range of f", i.e., $Hf\,;T\psi = Hf\,;T'\psi$.*

The requirement that a functor (like H in the theorem) preserves epis, is a mild one. In **Set** all polynomial functors preserve epis. Lehmann [9] argues that preservation of epis is an important property of functors.

We can now also see why homomorphisms do not preserve the validity of laws: outside the range of the homomorphism nothing can be inferred for the target algebra. (This may be a reason to consider the variety $V_F(E\mu F)$ instead of $\mathrm{Alg}(F, E)$, see Definition (16) and Theorem (17).) For example, imagine in **Set** the algebra $zero \triangledown add : \mathbf{1} + \mathbb{N} \to \mathbb{N}$ of finite naturals, where $zero$ is a neutral element for add. Form another algebra by adjoining a fictitious element ω to $\mathbb{N}$, and extend operation add as follows: for any natural x, $add'(x, \omega) = add'(\omega, x) = x$ and $add'(\omega, \omega) = \omega$. The injection of the original algebra into this new one is a homomorphism, but in the new algebra $zero$ is no longer neutral for add'.

Product For F-algebras ϕ, ψ (with carriers a and b say), and F-homomorphisms f, g with common source in $\mathrm{Alg}(F)$ we define

$$\begin{array}{lcl} \phi \times_F \psi & = & (F\pi_0 \,;\, \phi) \vartriangle (F\pi_1 \,;\, \psi) \quad : \quad F(a \times b) \to (a \times b) \\ f \vartriangle_F g & = & f \vartriangle g \\ \pi_{F,i} & = & \pi_i \,. \end{array}$$

It is then readily verified that $\times_F$, $\vartriangle_F$, $\pi_{F,i}$ form a categorical product in $\mathrm{Alg}(F)$, and we omit the subscript F since no confusion can result. In particular, $\pi_i : \phi_0 \times \phi_1 \to_F \phi_i$, i.e.,

$$(*) \quad \phi_0 \times \phi_1 \,;\, \pi_i \quad = \quad F\pi_i \,;\, \phi_i \,.$$

The generalisation to arbitrary products is straightforward; the proviso being that the base category has arbitrary products.

(14) Theorem $\mathrm{Alg}(F, E)$ *is closed under products.*

Proof We consider only binary products.

$$\begin{array}{ll} & T(\phi_0 \times \phi_1) = T'(\phi_0 \times \phi_1) \\ \Leftarrow & \quad \text{the projections are jointly monic in } \mathbf{C} \\ & T(\phi_0 \times \phi_1) \,;\, \pi_i = T'(\phi_0 \times \phi_1) \,;\, \pi_i \quad \text{for } i = 0, 1 \\ \equiv & \quad \textsc{Promotion} \text{ (condition is satisfied: see } (*) \text{ above) at both sides} \\ & H\pi_i \,;\, T\phi_i = H\pi_i \,;\, T'\phi_i \quad \text{for } i = 0, 1 \\ \Leftarrow & \quad \text{Leibniz, } \phi_i \text{ in } \mathrm{Alg}(F, E) \\ & \mathsf{true} \,. \end{array}$$

□

6 Initial algebras with laws

Let F be an endofunctor for which the initial algebra μF exists. Let E be a law of type $(F \to I) \to (H \to I)$ for some endofunctor H. As before, $\mathrm{Alg}(F, E)$ is the full subcategory of $\mathrm{Alg}(F)$ of algebras for which E holds. We are interested in an algebra that is initial in $\mathrm{Alg}(F, E)$; we shall denote it by $\mu(F, E)$.

Example (Trees cont'd) Take E to be the law that expresses both the neutrality of $\Box$ for $+\!\!+$, and the associativity of $+\!\!+$. Then $\beta = \Box' \triangledown \tau' \triangledown (+\!\!+') = \mu(F, E)$, if it exists, is the algebra of lists. Let $\phi := e \triangledown f \triangledown \oplus$ be another (F, E)-algebra. Then, by definition of initiality, the recursive equations

$$\begin{array}{lcl} \Box' \,;\, h & = & e \\ \tau' \,;\, h & = & f \\ +\!\!+' \,;\, h & = & \mathbb{I}h \,;\, \oplus \end{array}$$

have one and only one solution for h, denoted $(\!|\beta := \phi|\!) = (\!|\Box' \triangledown \tau' \triangledown (+\!\!+') := e \triangledown f \triangledown \oplus|\!)$. Actually, if the equations hold true for some $e, f, \oplus$, then we can *derive* that e is neutral for $\oplus$, and $\oplus$ is associative, at least on the range of h: see Lemma (13). □

To construct $\mu(F,E)$ when μF and E are given, we have to construct the least congruence p for algebra μF under which $T\mu F$ and $T'\mu F$ produce equivalent elements (when applied to an arbitrary argument), and then to construct the quotient algebra $\mu F/p$. We shall explain and define these notions categorically in the next subsection. Then, in the following subsection we shall discuss $\mu(F,E)$ and show the rôle of the PROMOTION property of transformers.

6.1 Induced congruence categorically

This subsection may be skipped without loss of continuity: we shall not really use the concepts defined here.

The category $a \downarrow \mathbf{C}$ of *morphisms under* a has as objects the morphisms (ranged over by $p, q, \ldots$) with source a, and $x : p \to_{a\downarrow\mathbf{C}} q$ means simply $p\,;x = q$. Similarly, the objects of $\mathbf{C} \downarrow\downarrow a$ are *parallel morphisms* (ranged over by $(f,g),(h,j),\ldots$) *with target* a, and $x : (f,g) \to_{\mathbf{C}\downarrow\downarrow a} (h,j)$ means $f = x\,;h \;\wedge\; g = x\,;j$. Further, for any category $\mathbf{A}$ and predicate P on the objects of $\mathbf{A}$, we let $\mathbf{A} \restriction P$ denote the full subcategory of objects of $\mathbf{A}$ satisfying P.

When $\mathbf{C} = \mathbf{Set}$, any (f,g) in $\mathbf{C} \downarrow\downarrow a$ represents a relation on a, namely the relation $\{(f\,z, g\,z) |\ z \text{ in the common source of } f \text{ and } g\} \subseteq a \times a$. We write $(f,g) \subseteq (h,j)$ if there exists an $x : (f,g) \to_{\mathbf{C}\downarrow\downarrow a} (h,j)$; indeed, in $\mathbf{Set}$ it means that the relation (f,g) on a is included in the relation (h,j).

When $\mathbf{C} = \mathbf{Set}$, any p in $a \downarrow \mathbf{C}$ represents an equivalence relation on a, namely the relation $\{(x,y) |\ px = py\} \subseteq a \times a$. We write $p \subseteq q$ if there exists an $x : p \to_{a\downarrow\mathbf{C}} q$; indeed, in $\mathbf{Set}$ it means that the equivalence relation p on a is included in the equivalence relation q. We also write $(f,g) \subseteq p$ if $f\,;p = g\,;p$ (i.e., $(f,g),p$ is a fork); again, in $\mathbf{Set}$ it means that the relation (f,g) is included in the equivalence relation p.

Given a relation (f,g) on a, p is called an equivalence relation *induced by* (f,g) if

(a) $\quad p$ is initial in $(a \downarrow \mathbf{C}) \restriction ((f,g) \subseteq \_)$

i.e., p is the $\subseteq$-smallest equivalence relation that includes (f,g). Categorically this notion is known as a coequaliser of (f,g). It is unique up to isomorphism.

Given an F-algebra $\phi : Fa \to a$, we call an equivalence relation p on a a *base-congruence for* ϕ if

(b) $\quad x;Fp = y\,;Fp \quad\Rightarrow\quad x\,;\phi\,;p = y\,;\phi\,;p$

i.e., 'componentwise' equivalent arguments x, y are mapped by ϕ to equivalent results. However, since we are primarily interested in algebras, another notion of congruence suggests itself: an equivalence relation p on a is called an *alg-congruence for* ϕ if

(c) $\quad p : \phi \to_F \psi$ for some ψ.

A relationship between the two notions is given by the following lemma.

(15) Lemma
(i) *p is alg-congruence for ϕ $\Rightarrow$ p is base-congruence for ϕ.*
(ii) *the converse of (i) is true if p has a pre-inverse.*

Proof (i) Let x, y be arbitrary. Then

$$\begin{array}{ll} & x\,;\phi\,;p = y\,;\phi\,;p \\ \equiv & \quad p \text{ is a homomorphism from } \phi, \text{ say } p : \phi \to_F \psi \\ & x\,;Fp\,;\psi = y\,;Fp\,;\psi \\ \Leftarrow & \quad \text{Leibniz} \\ & x\,;Fp = y\,;Fp \end{array}$$

as desired.
(ii) Let u be a pre-inverse of p, i.e., $u\,;p = id_a$. We show that p is a homomorphism.

$$\begin{array}{ll} & p : \phi \to_F (Fu\,;\phi\,;p) \\ \equiv & \quad \text{definition } \to_F \\ & \phi\,;p = Fp\,;Fu\,;\phi\,;p \\ \Leftarrow & \quad p \text{ is base-congruence for } \phi \text{ (taking } x, y := id, (Fp\,;Fu) \text{ in formula (b))} \\ & id\,;Fp = Fp\,;Fu\,;Fp \\ \equiv & \quad \text{functor, } u\,;p = id \\ & \mathsf{true}. \end{array}$$

Observe also that if $u\,;p = id = v\,;p$, then $Fu\,;\phi\,;p = Fv\,;\phi\,;p$, so that the target algebra of the homomorphism p is unique. □

Both notions of congruence lead to a notion of induced congruence. We leave the formulation of the base- variant to the reader. An equivalence relation p on a is called an *alg-congruence for ϕ induced by (f, g)* if

(d) p is initial in $(\phi \downarrow \mathrm{Alg}(F)) \restriction ((f, g) \subseteq \_)$

i.e., p is an alg-congruence for ϕ including (f, g), and is $\subseteq$-smallest among all such alg-congruences. It is unique up to isomorphism. The elegance of this definition (in comparison with that for the induced base-congruence) is the close analogy with (a). The analogy may be exploited in generalising a construction of an induced equivalence (coequaliser) to a construction of an induced alg-congruence. This has been done by Lehmann [9].

Given (f, g) and ϕ we denote the target algebra of the alg-congruence p for ϕ induced by (f, g) as $\phi/(f, g)$, i.e.,

$$p : \phi \to_F \phi/(f, g).$$

For category **Set** the construction of p and $\phi/(f,g)$ is well known. Lehmann [9] gives a general, abstract construction for the induced alg-congruence, for various categories of algebras with partially ordered sets as carrier (with and without a least element, and monotonic or continuous functions as morphisms, for several kinds of continuity). It is unfortunately not clear to us whether his construction specialises to the well-known **Set**-theoretical construction (e.g., by taking the trivial order on all sets), and what precisely the conditions on the category are. Lehmann's construction does require that functor F preserves epis, and that free F-algebras (something like $\mu(\underline{a} + F)$) exist.

Remark We have devised a simple explicit construction of the induced base-congruence (simulating the construction in **Set**) for any ω-cocontinuous functor F, and any ω-cocontinuous category that has all finite coequalisers, pushouts, and kernel pairs (pullbacks of equal pairs), assuming that the kernel pair functor is ω-cocontinuous. But, alas, the proof is too complicated to present here. □

6.2 Initial lawful algebra

Throughout the subsection let F and $E = (T, T') : (F \to I) \to (H \to I)$ be declared globally, and assume that $\alpha = \mu F$ exists. We start by summarising a result of Lehmann [8, 9].

(16) Definition *For a pair (f,g) of morphisms with common source and the carrier of α as their common target, the (f,g)-variety $V_F(f,g)$ is the full subcategory of* Alg(F) *of algebras ϕ for which $f \,;(\!|\phi|\!) = g \,;(\!|\phi|\!)$.*

(17) Theorem (Lehmann [9]) *For $\mathbf{C} = \mathbf{Set}$, and for any $\mathbf{C}$ that satisfies the conditions of Lehmann's theorem, such as several order-enriched categories, and assuming that F preserves epis, any variety $V_F(f,g)$ has an initial object, denoted $\alpha/(f,g)$. Moreover $(\!|\alpha/(f,g)|\!)$ is epic in $\mathbf{C}$.*

Catamorphism $(\!|\alpha/(f,g)|\!)$ is, in the terminology of the previous subsection, the '*alg-congruence for α induced by (f,g)*'. Its type is, of course, $(\!|\alpha/(f,g)|\!) : \alpha \to_F \alpha/(f,g)$.

The following result enables us to exploit Lehmann's theorem.

(18) Theorem *Suppose that H preserves epis. Then* Alg(F,E) *is a full subcategory of $V_F(E\alpha)$, and contains each ϕ of $V_F(E\alpha)$ for which $(\!|\phi|\!)$ is epic in $\mathbf{C}$.*

Proof Since both Alg(F,E) and $V_F(E\alpha)$ are full subcategories of Alg(F), we have to show, for the first claim, that any ϕ in Alg(F,E) is also in $V_F(E\alpha)$. For this we argue

$$
\begin{array}{ll}
 & T\alpha \,;(\!|\phi|\!) = T'\alpha \,;(\!|\phi|\!) \\
\equiv & \quad \text{PROMOTION (condition is satisfied: } (\!|\phi|\!) : \alpha \to_F \phi) \text{ at both sides} \\
 & H(\!|\phi|\!) \,; T\phi = H(\!|\phi|\!) \,; T'\phi \\
\Leftarrow & \quad \text{Leibniz, } E\phi \text{ holds} \\
 & \mathsf{true}\,.
\end{array}
$$

For the second claim let ϕ be in $V_F(E\alpha)$ be such that $(\!|\phi|\!)$ is epic in $\mathbf{C}$. We show that $E\phi$ holds, so that ϕ is in $\mathrm{Alg}(F,E)$ as well.

$$
\begin{array}{ll}
 & T\phi = T'\phi \\
\equiv & \quad (\!|\phi|\!) \text{ is epic, } H \text{ preserves epis} \\
 & Hp\,;T\phi = Hp\,;T'\phi \\
\equiv & \quad \textsc{Promotion} \text{ (condition is satisfied: } F(\!|\phi|\!)\,;\phi = \alpha\,;(\!|\phi|\!)) \text{ at both sides} \\
 & T\alpha\,;(\!|\phi|\!) = T'\alpha\,;(\!|\phi|\!) \\
\equiv & \quad \text{definition (16), } \phi \text{ is in } V_F(E\alpha) \\
 & \mathsf{true}\,.
\end{array}
$$

□

(19) Corollary *Suppose both F and H preserve epis, and $\mathbf{C}$ satisfies the conditions of Lehmann's theorem [9], see (17). Then $\mathrm{Alg}(F,E)$ has an initial object, denoted $\mu(F,E)$.*

Proof By Theorem (17) $V_F(E\alpha)$ has an initial object $\alpha/E\alpha$, and $(\!|\alpha/E\alpha|\!)$ is epic in $\mathbf{C}$. (Here the conditions on F and $\mathbf{C}$ are used.) Then by Theorem (18) $\alpha/E\alpha$ is also in $\mathrm{Alg}(F,E)$, and moreover it is also initial in $\mathrm{Alg}(F,E)$. (Here it is used that H preserves epis.) □

* * *

Although the following is independent of the precise nature of laws and transformers, we cannot resist the temptation to include it. The theorem is useful for the development of efficient programs, as we shall explain afterwards. We put $\alpha = \mu F$ and $\beta = \mu(F,E)$.

(20) Theorem (Meertens) *Suppose (α and also) β exists, and suppose $(\!|\alpha := \beta|\!)$ has a pre-inverse u. Then*

$$
\begin{array}{lcl}
(\!|\beta := \phi|\!) & = & u\,;(\!|\alpha := \phi|\!) \\
(\!|\alpha := \phi|\!) & = & (\!|\alpha := \beta|\!)\,;u\,;(\!|\alpha := \phi|\!)
\end{array}
$$

for any ϕ in $\mathrm{Alg}(F,E)$.

Proof For the first claim we argue

$$
\begin{array}{ll}
 & (\!|\beta := \phi|\!) \;=\; u\,;(\!|\alpha := \phi|\!) \\
\equiv & \quad u \text{ is pre-inverse of } (\!|\alpha := \beta|\!) \\
 & u\,;(\!|\alpha := \beta|\!)\,;(\!|\beta := \phi|\!) \;=\; u\,;(\!|\alpha := \phi|\!) \\
\Leftarrow & \quad \text{Leibniz} \\
 & (\!|\alpha := \beta|\!)\,;(\!|\beta := \phi|\!) \;=\; (\!|\alpha := \phi|\!) \\
\equiv & \quad \text{cata-}\textsc{Fusion} \\
 & (\!|\beta := \phi|\!) : \beta \to_F \phi \\
\equiv & \quad \text{cata-}\textsc{Self} \\
 & \mathsf{true}\,.
\end{array}
$$

The second claim is an immediate corollary:

$$
\begin{array}{ll}
& (\!|\alpha := \beta|\!)\,;u\,;(\!|\alpha := \phi|\!) \\
= & \quad \text{just shown} \\
& (\!|\alpha := \beta|\!)\,;(\!|\beta := \phi|\!) \\
= & \quad \text{cata-FUSION (condition is satisfied: } (\!|\beta := \phi|\!) : \beta \to_F \phi) \\
& (\!|\alpha := \phi|\!)\,.
\end{array}
$$

The existence of β (hence the well definedness of $(\!|\beta := \_|\!)$) is guaranteed by the previous theorems if F and H preserve epis. □

In **Set**, the carrier of β consists of the $\simeq$-equivalence classes of the carrier of α, where $\simeq$ is the least equivalence relation that contains all pairs $(T\alpha)z, (T'\alpha)z$ for all z. A pre-inverse u of $(\!|\alpha := \beta|\!)$ chooses for any equivalence class a representative in the class. So the theorem allows you to compute $(\!|\beta := \phi|\!)$ at x by computing $(\!|\alpha := \phi|\!)$ at a representative of x instead. In this way you may improve the operational efficiency of a program.

Example (Trees cont'd) Let $E(e \triangledown id \triangledown \oplus)$ express that $\oplus$ is an associative binary operation with neutral element e, and suppose that the law holds for $e \triangledown id \triangledown \oplus$. The value of $(\!|\beta := e \triangledown id \triangledown \oplus|\!)$ at arguments $x +\!\!+' (y +\!\!+' z)$ and $(x +\!\!+' y) +\!\!+' z$ is

$$
\begin{array}{lcl}
(\!|\beta := e \triangledown id \triangledown \oplus|\!)(x +\!\!+' (y +\!\!+' z)) & = & x \oplus (y \oplus z) \\
(\!|\beta := e \triangledown id \triangledown \oplus|\!)((x +\!\!+' y) +\!\!+' z) & = & (x \oplus y) \oplus z\,.
\end{array}
$$

Due to associativity both results are the same, yet the computations as suggested by the right hand sides may differ operationally. For example, the first alternative is more efficient if $x \oplus y$ takes time linear in $size\,x$, and $size(x \oplus y) = size\,x + size\,y$. (This is valid in most functional programming languages for $\oplus$ equal to the concatenation of lists.) Thus associativity may be exploited. More generally, let u be the function that sends $x +\!\!+' y +\!\!+' \ldots +\!\!+' z$ (with any parenthesation) to $x +\!\!+ (y +\!\!+ (\ldots +\!\!+ z))$ (with parenthesation to the right). The theorem asserts that $(\!|\beta := e \triangledown id \triangledown \oplus|\!) = u\,;(\!|\alpha := e \triangledown id \triangledown \oplus|\!)$, and by the argument above we know that the catamorphism in the right hand side is more efficient than that in the left hand side. (It is quite easy to express u explicitly. But possibly u disappears completely, namely when this transformation is but one step in a large program transformation.) □

In a similar way the second claim of the theorem asserts that if ϕ satisfies E, then "within the argument" of $(\!|\phi|\!)$ you may act as if α satisfies E. More precisely, writing $(\!|\alpha := \ldots|\!)$ as $(\!|\ldots|\!)$, the theorem asserts that $(\!|\phi|\!)$ makes $(\!|\beta|\!)\,;u$ and id equal (coequalises them); and we have:

$$
(\!|\beta|\!)\,;u \text{ equals } id \quad \Rightarrow \quad \alpha \text{ satisfies } E\,.
$$

This is shown as follows.

$T\alpha = T'\alpha$

$\equiv$ premise: $(\!|\beta|\!)$ has post-inverse

$T\alpha \,;\, (\!|\beta|\!) = T'\alpha \,;\, (\!|\beta|\!)$

$\equiv$ PROMOTION (condition is satisfied: $(\!|\beta|\!) : \alpha \to_F \beta$) at both sides

$H(\!|\beta|\!) \,;\, T\beta = H(\!|\beta|\!) \,;\, T'\beta$

$\equiv$ Leibniz, $E\beta$ holds

true.

(The converse implication does hold too, as the reader may wish to verify.)

7 Any colimit is an initial lawful algebra

Lambert Meertens has made the following observation. Given an arbitrary colimit we can construct an endofunctor F and a law E such that (the junc of) the colimit is an initial (F, E)-algebra, provided the category has arbitrary coproducts. This is another evidence for the expressiveness of our notion of law. We shall first perform the construction for coequalisers, and then for colimits in general.

Let f, g be a parallel pair with target a, and let p be a coequaliser of f, g. This means, by definition, that for any q with $f \,;\, q = g \,;\, q$ there exists a morphism, which we denote $p\backslash q$, such that

$$p \,;\, x = q \quad \equiv \quad x = p\backslash q \qquad \text{coequaliser-CHARN}$$

Now take $F = \underline{a}$, the constant functor mapping any morphism onto id_a. Take $E = (T, T')$ with $Tq = f \,;\, q$ and $T'q = g \,;\, q$ for any $q : Fb \to b \; (= a \to b)$. We observe $T = \underline{f}; I$ (in the notation of Theorem (6)), and similarly $T' = \underline{g}; I$, and so E is a law (by that same theorem). Further, Ep holds, and $p : F(\mathrm{tgt}\, p) \to \mathrm{tgt}\, p$. So p is an (F, E)-algebra. To show initiality we shall prove cata-CHARN, deriving along the way a definition for the required $(\!|p := q|\!)$.

$x : p \to_F q$

$\equiv$ definition $\to_F$ and our F

$p \,;\, x = q$

$\equiv$ coequaliser-CHARN, noting that $f \,;\, q = g \,;\, q$

$x = p\backslash q$

$\equiv$ **defining** $(\!|p := q|\!) = p\backslash q$

$x = (\!|p := q|\!)$.

Now we generalise the above construction to arbitrary colimits. The p and q above become cocones γ, δ or algebras γ', δ' below, the f and g become (the arrows in) the diagram D, and law E is going to express "the commutativity of all triangles".

Let D be a diagram in $\mathbf{C}$. A cocone for D is a family $\delta = (a \text{ in } D :: \delta_a)$ such that

(a) $\quad \forall f : a \to b \text{ in } D :: \quad \delta_a = f \,;\, \delta_b \,.$

A cocone γ is a colimit for D if for any cocone δ for D there exists a morphism, which we denote $\gamma \backslash \delta$, such that

(b) $\quad \forall(a \text{ in } D :: \quad \gamma_a \,;\, x = \delta_a) \quad \equiv \quad x = \gamma \backslash \delta$ $\qquad$ colimit-CHARN

Now take $F = \underline{\Sigma D}$, where ΣD = (the carrier of) the coproduct of all objects in D. Similarly to the Trees example any F-algebra $\delta : \Sigma D \to d$ can be written as $\delta = \nabla(a \text{ in } D :: \delta_a)$ where δ_a abbreviates $inj_a \,;\, \delta$. We design E such that $E\delta$ equivales (a) above:

$$\begin{array}{lcl} E & = & \text{the conjunction of the collection } (f : a \to b \text{ in } D :: \quad (T_a, T'_{f,b})) \\ T_a\delta & = & \delta_a \quad (= inj_a \,;\, \delta) \\ T'_{f,b}\delta & = & f \,;\, \delta_b \quad (= f \,;\, inj_b \,;\, \delta) \,. \end{array}$$

Indeed, $\delta = \nabla(a :: \delta_a)$ is an (F, E)-algebra iff $\delta' = (a :: \delta_a)$ is a cocone for D. Moreover, by Theorem (6) T_a and $T'_{f,b}$ are transformers, so that $(T_a, T'_{f,b})$ is a law, and by Theorem (10) the conjunction E can be expressed as a law.

Let $\gamma = (a :: \gamma_a)$ be a colimit for D. We claim that $\gamma' = \nabla(a :: \gamma_a)$ is initial in $\mathrm{Alg}(F, E)$. To show this let $\delta' = \nabla(a :: \delta_a)$ be any (F, E)-algebra. Then, as argued above, $\delta = (a :: \delta_a)$ is a cocone for D, so $\gamma \backslash \delta$ satisfying (b) exists. It is now readily shown that $\gamma \backslash \delta$ taken as $(\!|\gamma' := \delta'|\!)$ meets the requirement cata-CHARN:

$$\begin{array}{ll} & x : \gamma' \to_F \delta' \\ \equiv & \quad \text{definition } \to_F \\ & \gamma' \,;\, x = Fx \,;\, \delta' \\ \equiv & \quad \text{definition } F \text{ as a constant functor} \\ & \gamma' \,;\, x = \delta' \\ \equiv & \quad \text{definition } \gamma' \text{ and } \delta' \text{ and coproduct} \\ & \forall a \text{ in } D :: \quad \gamma_a \,;\, x = \delta_a \\ \equiv & \quad \text{colimit-CHARN} \\ & x = \gamma \backslash \delta \\ \equiv & \quad \text{definition } `(\!|\gamma' := \delta'|\!)' \\ & x = (\!|\gamma' := \delta'|\!) \,. \end{array}$$

Thus γ' is initial indeed.

8 Equational specification of datatypes

A datatype like *stack* with operations *empty*, *push*, *isempty*, *top* and *pop* has not the form of an algebra $\phi : Fa \to a$, but is rather a pair (ϕ, ψ) with $\phi : Fa \to a$ and $\psi : a \to Ga$, for some endofunctors F, G. To be specific, for *stack* we have

$$\begin{array}{lcllcl} \phi & = & empty \triangledown push & : & 1 + b \times a \to a & = & Fa \to a \\ \psi & = & isempty \vartriangle top \vartriangle pop & : & a \to \mathbb{B} \times b \times a & = & a \to Ga \end{array}$$

where b is the type of the stacked values (and a is the type of the stacks themselves). We call such a pair (ϕ, ψ) a (single-sorted) F, G-bi-algebra. Often some law E is imposed that "defines" *isempty*, *top*, and *pop* (the ψ-part) in terms of *empty* and *push* (the ϕ-part). For *stack* we usually find

$$\begin{array}{lcl} empty\,;isempty & = & true \\ push\,;isempty & = & false \\ push\,;top & = & \pi_0^2 \\ push\,;pop & = & \pi_1^2 . \end{array}$$

Written as two equations:

$$\begin{array}{lcl} empty\,;\psi & = & id_1\,;true \vartriangle \ldots \vartriangle \ldots \\ push\,;\psi & = & id_b \times \psi\,;(!\,;false) \vartriangle \pi_0^2 \vartriangle (\pi_1^2\,;\pi_1^3 \vartriangle \pi_2^3\,;push) \end{array}$$

where on the dots there have to be expressions of type $1 \to b$ and $1 \to a$ respectively, defining the top and pop of an empty stack. (It is outside the scope of this paper to discuss this aspect in detail.) As usual we can even combine the two equations into one, thus obtaining a law

$$E(\phi, \psi) \quad = \quad \text{“}\ \phi\,;\psi \ = \ F\psi\,;T\phi\ \text{”}$$

for some transformer T of type $(F \to I) \to (I \to G)$. The theorem below asserts that with such a law the datatype (initial bi-algebra) (ϕ, ψ) is isomorphic to the initial F-algebra (the ϕ-part) to which $(\!|T\phi|\!)$ (the ψ-part) is added as a derived operation. Since for the F above the initial F-algebra is known as the cons-lists, we find that the datatype *stack* is semantically just the algebra of cons-lists with some additional derived operations, "destructors" in this case.

Formalisation Let F, G be endofunctors. An F, G-bi-algebra is a pair (ϕ, ψ) with $\phi : Fa \to a$ and $\psi : a \to Ga$, for some a called the carrier of the bi-algebra. (So, the pair consists of an algebra and a co-algebra with the same carrier.) Given F, G-bi-algebras (ϕ, ψ) and (χ, ω), we say f is a homomorphism from (ϕ, ψ) to (χ, ω), denoted $f : (\phi, \chi) \to^F_G (\chi, \omega)$, if $f : \phi \to_{F,I} \chi$ and $f : \psi \to_{I,G} \omega$ (using the notation of di-algebras of Section 2). This determines a category $\mathrm{BiAlg}(F, G)$: the objects are F, G-bi-algebras, the

morphisms are F, G-bi-homomorphisms, and the composition and identities are inherited from the base category[3].

It is straightforward to extend the notion of transformer and law in such a way that they work on bi-algebras; the type takes the form $((F \to I), (I \to G)) \to (H \to J)$. If E is such a law we let $\mathrm{BiAlg}(F, G; E)$ denote the full subcategory of $\mathrm{BiAlg}(F, G)$ containing all and only bi-algebras that satisfy E.

(21) Theorem *Let T be a transformer of type $(F \to I) \to (FG \to G)$, and suppose that $\alpha = \mu F$ exists. Let E be the law suggested by*

$$E(\phi, \psi) \quad = \quad \text{“ } \phi ; \psi \;=\; F\psi ; T\phi \text{ ”} \qquad \text{for } F, G\text{-bialgebra } (\phi, \psi).$$

Then $(\alpha, (\!|T\alpha|\!))$ is initial in $\mathrm{BiAlg}(F, G; E)$.

Proof (Observe that law E is well-formed; the type of both sides of the equation is $Fa \to Ga$ where a is the carrier of the argument.) Let (ϕ, ψ) be any F, G-bi-algebra for which E holds. We shall show that

$$x : (\alpha, (\!|T\alpha|\!)) \to_G^F (\phi, \psi) \qquad \equiv \qquad x = (\!|\phi|\!)$$

thus establishing the existence and uniqueness of an F, G-bi-homomorphism (namely $(\!|\phi|\!)$) from $(\alpha, (\!|T\alpha|\!))$ to (ϕ, ψ).

$$\begin{array}{ll} & x : (\alpha, (\!|T\alpha|\!)) \to_G^F (\phi, \psi) \\ \equiv & \quad \text{definition } \to_G^F \\ & x : \alpha \to_F \phi \;\wedge\; x : (\!|T\alpha|\!) \to_{G,I} \psi \\ \equiv & \quad \text{cata-SELF} \\ & x = (\!|\phi|\!) \;\wedge\; (\!|\phi|\!) : (\!|T\alpha|\!) \to_{G,I} \psi \\ (*) \quad \equiv & \quad \text{below} \\ & x = (\!|\phi|\!). \end{array}$$

It remains to justify step $(*)$. For this we argue

$$\begin{array}{ll} & (\!|\phi|\!) : (\!|T\alpha|\!) \to_{G,I} \psi \\ \equiv & \quad \text{definition } \to_{G,I} \\ & (\!|T\alpha|\!) ; G(\!|\phi|\!) = (\!|\phi|\!) ; \psi \\ \equiv & \quad \text{rhs: cata-FUSION (condition '}\psi : \phi \to_F T\phi\text{' follows from } E(\phi, \psi)) \\ & (\!|T\alpha|\!) ; G(\!|\phi|\!) = (\!|T\phi|\!) \\ \Leftarrow & \quad \text{cata-FUSION} \end{array}$$

[3] Using product categories one may define bi-algebras as di-algebras: $\mathrm{BiAlg}(F, G) = \mathrm{DiAlg}(F \vartriangle I, I \vartriangle G)$. Similarly for transformers for such algebras. Hence there are no new concepts involved, but only specialisations of known ones.

$$
\begin{array}{ll}
 & T\alpha\,;G(\!|\phi|\!) = FG(\!|\phi|\!)\,;T\phi \\
\Leftarrow & \quad \text{PROMOTION} \\
 & \alpha\,;(\!|\phi|\!) = F(\!|\phi|\!)\,;\phi \\
\equiv & \quad \text{cata-SELF} \\
 & \mathsf{true}\,.
\end{array}
$$

□

Similarly one may specify an $F+G$-algebra $\phi \triangledown \psi$ by forcing the ψ-part to be determined by the ϕ-part. In this case ψ is an additional derived operation that is a "constructor", like ϕ.

(22) Theorem *Let T be a transformer of type $(F \to I) \to (G \to I)$, and suppose that $\alpha = \mu F$ exists. Let E be the law suggested by*

$$E(\phi \triangledown \psi) \quad \equiv \quad \text{“}\ \psi \ = \ T\phi\ \text{”}\,.$$

Then $\alpha \triangledown T\alpha$ is initial in $\mathrm{Alg}(F+G;E)$.

Proof We show initiality of $\alpha \triangledown T\alpha$ by establishing cata-CHARN. Let $\phi \triangledown \psi$ be an arbitrary $F+G$-algebra for which E holds. Then

$$
\begin{array}{ll}
 & x:\ \alpha \triangledown T\alpha\ \to_{F+G}\ \phi \triangledown \psi \\
\equiv & \quad \text{definition } \to_{F+G} \\
 & x : \alpha \to_F \phi\ \ \wedge\ \ x : T\alpha \to_G \psi \\
\equiv & \quad \text{cata-SELF} \\
 & x = (\!|\phi|\!)\ \ \wedge\ \ (\!|\phi|\!) : T\alpha \to_G \psi \\
(*) \quad \equiv & \quad \text{below} \\
 & x(\!|\phi|\!)\,.
\end{array}
$$

Step $(*)$ is verified as follows.

$$
\begin{array}{ll}
 & (\!|\phi|\!) : T\alpha \to_G \psi \\
\equiv & \quad \text{law } E \text{ holds for } \phi \triangledown \psi \\
 & (\!|\phi|\!) : T\alpha \to_G T\phi \\
\Leftarrow & \quad \text{PROMOTION} \\
 & (\!|\phi|\!) : \alpha \to_F \phi \\
\equiv & \quad \text{cata-SELF} \\
 & \mathsf{true}\,.
\end{array}
$$

□

It is straightforward to combine both theorems, and generalise it to the case of triples (ϕ, ψ, χ) where ϕ is an F, G-dialgebra, ψ is an H-algebra, and χ is a J-co-algebra, all with the same carrier, and ψ, χ being determined in terms of ϕ by means of a law.

* * *

The above exposition should be compared with current literature on 'equational specification of datatypes' such as Ehrig and Mahr's book [4]. Our formalism above is entirely directed to the semantics (of algebras, or datatypes), whereas signatures and other syntactic aspects are prominently present in Ehrig and Mahr's formalism. As a result, even in the discussions of purely semantic aspects they are forced to take into account irrelevant aspects like scope rules —appearing in the decision to incorporate a parameter algebra into the result algebra— and sharing of implementations —appearing in the notion of persistency— and so on. This gives a lot of unnecessary junk and confusion, and such a treatise is in no way initial. Thanks to our notion of transformer and law we are able to avoid the introduction of non-semantic aspects.

9 Conclusion

We have proposed a semantical, categorical, characterisation of what a term (as used in conditional equations) is: the PROMOTION property. The property is almost as simple as the defining property of functor, and a mapping that satisfies the PROMOTION property is called 'transformer'. The reasonability of the proposal has been shown by various theorems on the expressiveness of transformers. The simplicity of various proofs dealing with laws is further evidence of the success of the notion of transformer.

We have also sketched a great simplification of the theory of equational specification of datatypes as far as only semantic aspects are concerned. Much more in this area can be done.

Thanks to the formalisation of the notion of law, one can now formulate conjectures, statements and proofs about them in general. For example, since long there was a feeling that so-called lifting preserves the validity of all "algebraic" laws; so a lifted commutative operation would be commutative again. Recently, Meertens and Van der Woude have been able to formally prove this conjecture — using the notion of transformer.

Acknowledgement The Constructive Algorithmics Club at Utrecht University provided a stimulating forum to present and discuss the work reported here. In particular I have had useful discussions with Jaap van der Woude and Lambert Meertens; the latter suggested the contents of Section 7 and part of Theorem (20). Erik Meijer brought Lehmann [8] to my attention. Ross Paterson suggested several improvements in the presentation, and also some further generalisations [some of which still have to be incorporated in the paper].

References

[1] R.C. Backhouse. An exploration of the Bird-Meertens formalism. Technical Report CS8810, Dept of Math and Comp Sc, University of Groningen, 1988.

[2] R.S. Bird. The promotion and accumulation strategies in transformational programming. *ACM Transactions on Programming Languages and Systems*, 6(4):487–504, 1984. Addendum: Ibid. 7(3):490–492, 1985.

[3] J. Darlington. A synthesis of several sorting algorithms. *Acta Informatica*, 11(1):1–30, 1978.

[4] H. Ehrig and B. Mahr. *Fundamentals of Equational Specification 1 — equations and initial semantics.* Springer Verlag, 1985.

[5] M.M. Fokkinga. Calculate categorically! Technical Report CS-R9132, CWI, Amsterdam, Amsterdam, June 1991.

[6] M.M. Fokkinga and E. Meijer. Program calculation properties of continuous algebras. Technical Report CS-R9104, CWI, Amsterdam, Amsterdam, January 1991.

[7] T. Hagino. *Category Theoretic Approach to Data Types.* PhD thesis, University of Edinburgh, 1987.

[8] D.J. Lehmann. On the algebra of order — extended abstract. In *19th Symposium on the Foundations of Computer Science (FOCS)*, pages 214–220. IEEE, 1978.

[9] D.J. Lehmann. On the algebra of order. *Journal of Computer and System Sciences*, 21:1–23, 1980.

[10] D.J. Lehmann and M.B. Smyth. Algebraic specification of data types: A synthetic approach. *Math. Systems Theory*, 14:97–139, 1981.

[11] G. Malcolm. *Algebraic Data Types and Program Transformation.* PhD thesis, Groningen University, Netherlands, 1990.

[12] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14(2–3):255–280, September 1990.

[13] E.G Manes and M.A. Arbib. *Algebraic Approaches to Program Semantics.* Text and Monographs in Computer Science. Springer Verlag, 1986.

[14] L. Meertens. Algorithmics — towards programming as a mathematical activity. In J.W. de Bakker and J.C. van Vliet, editors, *Proceedings of the CWI Symposium on Mathematics and Computer Science*, pages 289–334. North-Holland, 1986.

[15] L. Meertens. Constructing a calculus of programs. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*, Lect. Notes in Comp. Sc., pages 66–90. Springer Verlag, 1989. LNCS 375.

[16] B.C. Pierce. A taste of category for computer scientist. *ACM Computing Surveys.* To appear. Also Tech Report CMU-CS-90-113, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 125213.

[17] P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pages 347–359. ACM Press, September 1989. FPCA '89, Imperial College, London.

[18] J.C.S.P. van der Woude. C4, general manipulative nonsense? Tech Report in preparation, CWI, Amsterdam, March 1990.