Math. Struct. in Comp. Science (1993), *vol.* 11, *pp.* 1–000 Copyright © Cambridge University Press

Datatype Laws without Signatures

Maarten M. Fokkinga[†]

University of Twente, dept. INF, P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: fokkinga@cs.utwente.nl

Received 1991

Using the well-known categorical notion of 'functor' one may define the concept of datatype (algebra) without being forced to introduce a signature, that is, names and typings 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.
The categorical notion of 'transformer' developed in this paper allows the same approach to laws: without using signatures one can define the concept of law for datatypes (lawful algebras), and investigate the equational specification of datatypes in a syntax-free way. A transformer is a special kind of functor and also a natural transformation on the level of dialgebras. Transformers are quite expressive, satisfy several closure properties, and are related to naturality and Wadler's Theorems For Free. In fact, any colimit is an initial lawful algebra.

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, transformer.

A. Introduction

1 *The problem.* Most mathematical formalisations of the intuitive notion of 'datatype' 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, with the notion of 'sketch' as described by Barr and Wells (1990). A signature gives the syntactic appearance of the algebra; it gives the names of the sorts (types), the names and arities of the operations and constants, and for each operation a syntactic indication of the types of its arguments and result. 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, for example when investigating the existence of certain kinds of algebras, or the

[†] Research partly done while at the CWI, Amsterdam.

(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 characterises the type structure of an algebra without naming 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 for the operations of an 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 (1991, remark 2.2.3) explicitly says: "The framework has apparently never been extended to include algebras with equations."

2 *The solution.* We shall propose a categorical description of 'law' that avoids naming, and is of the same simplicity as the definition of 'functor'. To be specific, each of the two terms of an equation shall be just a mapping T, from (di)algebras to (di)algebras, that satisfies a particular so-called TRANSFORMER property; and such a T is called a *transformer*. A transformer is a special kind of functor, as well as a natural transformation.

There are several properties that transformers should have if the notion is to be of any use. So, in Section D we show that transformers can be composed in various ways to form transformers again, and are thus as expressive as the usual syntactic terms in conditional equations. In addition, 'laws' are shown to be closed under conjunction. Moreover, in that section we also explore the notion of transformer by showing some relation between transformers and Wadler (1989)'s Theorems For Free theorem and naturality. In Section E we give, for each law E, conditions under which the class $\mathcal{A}lg(F,E)$ of F-algebras satisfying E is closed under subalgebras, product algebras and homomorphic images. In Section F we give conditions under which $\mathcal{A}lg(F,E)$ has an initial object, the initial F,E-algebra. In the same section we also show how to exploit a law E of the initial F,E-algebra in programming. In Section G we show that any colimit is in fact an initial F,E-algebra for some suitable choice of F and E. And finally, in Section H, we show the transformers in action in the theory of equational specification of datatypes, by proving two little theorems concerning the isomorphy of two differently specified datatypes.

The simplicity of the proofs of the claims above demonstrates the success of our formalisation.

B. Preliminaries

Apart from giving our notational conventions, we give in this section 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 (1981), Malcolm (1990a;

1990b), and Fokkinga (1992a; 1992b). Also Manes and Arbib (1986) and Pierce (1991) give a brief description.

Nomenclature. Variables $\mathcal{A}, \mathcal{B}, \mathcal{C}$ vary over categories; $a, b, c, \ldots$ vary over objects; f, g, h, j, …, p, q, …, α, β, φ, ψ, … vary over morphisms (actually, φ, ψ, .. vary over algebras and α, β over initial algebras); F, G, H, J, K, … vary over functors, U varies over forgetful functors, and μF shall denote a distinguished initial F-algebra, assuming it exists. The unit type, a fixed terminal object, is denoted $\mathit{1}$. We use x, y, z to denote various entities. Formula $x \colon a \to_{\mathcal{A}} b$ means: x is a morphism in $\mathcal{A}$ with source a and target b ($\mathrm{src}_{\mathcal{A}} x = a$ and $\mathrm{tgt}_{\mathcal{A}} x = b$). 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 Id ; functor $\mathit{Id} \times \mathit{Id}$ is abbreviated to $\mathit{I\!I}$, so that $\mathit{I\!I} x = x \times x$. A constant function mapping each argument onto x is denoted $\underline{x}$. The notation $\underline{x}$ is also used for the constant functor, mapping objects onto x and morphisms onto id_x .

If a category has been declared the *default* one, then we shall mostly suppress mentioning the category, certainly where it occurs as a subscript. Unless stated explicitly otherwise, a morphism is a morphism in the default category, and similarly for objects, source and target, and endofunctors. We denote composition of the default category (and all the categories that inherit the composition of the default category) in diagrammatic order: if $x \colon a \to b$ and $y \colon b \to c$ then $x \mathbin{;} y \colon a \to c$.

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

Product and coproduct. In the examples and at several other places we assume that the default category has products and coproducts. For products (in $\mathcal{S}et$: cartesian products) we use the notations $\times$, exl, exr , and $\vartriangle$ (and $\mathit{ex}_{n,i}$ for n-ary products in the examples). Here $\mathit{exl}/\mathit{exr}$ is the extraction of the left/right coordinate, and $f \vartriangle g$ (pronounced: f `con` g) is usually denoted $\langle f, g \rangle$; in $\mathcal{S}et$ we have $(f \vartriangle g)x = (f\,x, g\,x)$. For coproducts (in $\mathcal{S}et$: disjoint unions) we use the notation $+$, inl, inr , and $\triangledown$ (and $\mathit{in}_{n,i}$ in the examples). The usual notation for $f \triangledown g$ (pronounced: f `dis` g) is $[f, g]$; in $\mathcal{S}et$ we have $(f \triangledown g)(\mathit{inl}\, x) = f\,x$, and so on. So, the typing reads

$f \vartriangle g$	:	$a \to b \times c$	whenever $f \colon a \to b$ and $g \colon a \to c$
$f \times g$	:	$a \times b \to c \times d$	whenever $f \colon a \to c$ and $g \colon b \to d$
exl	:	$a \times b \to a$	
exr	:	$a \times b \to b$	
$f \triangledown g$	:	$a + b \to c$	whenever $f \colon a \to c$ and $g \colon b \to c$

$f + g$	:	$a + b \to c + d$	whenever $f\colon\ a \to c$ and $g\colon\ b \to d$
inl	:	$a \to a + b$	
inr	:	$b \to a + b\,.$	

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

We also use $\mathit{Exl}, \mathit{Exr}$ to denote the extraction functors from a product category to the component categories.

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

4 *Running example: trees.* In this paper we use the well-known algebra of *binary structures* (abbreviated to *trees*) over the set nat as an example. We could be more general, and consider trees over an arbitrary set a, but we wish to avoid the extra notions and notation involved.

The default category is $\mathcal{S}et$. The carrier of the algebra is the set tree that consists of all finite binary structures with values from nat at the tips. There are three functions nil, tip, and join:

nil	:	$\mathit{1} \to \mathit{tree}$	the nil structure
tip	:	$\mathit{nat} \to \mathit{tree}$	the tip former
join	:	$\mathit{II}\,\mathit{tree} \to \mathit{tree}$	the join operation, joining two structures.

The notation $x \,\mathrm{join}\, y$ will be used as an alternative for $\mathit{join}(x, y)$. Put

$$\begin{aligned} \alpha &= \mathit{nil} \triangledown \mathit{tip} \triangledown \mathit{join} \\ \mathit{OneNatBin} &= \underline{\mathit{1}} + \underline{\mathit{nat}} + \mathit{II}, \end{aligned}$$

that is,

$$\mathit{OneNatBin}\, x = \mathit{id}_{\mathit{1}} + \mathit{id}_{\mathit{nat}} + x \times x \quad \text{for each function } x\,.$$

Then $\alpha\colon\ \mathit{OneNatBin}\,\mathit{tree} \to \mathit{tree}$, so α is an $\mathit{OneNatBin}$-algebra with carrier tree. We shall see in paragraph 6 how to define the algebra formally. In paragraph 15 we shall see that these binary structures are effectively lists, bags, or sets when operation join and nil satisfy suitable laws.

Another $\mathit{OneNatBin}$-algebra is $\underline{0} \triangledown \underline{1} \triangledown (+)$; this one has carrier nat. In the examples to come we let $e \triangledown f \triangledown \oplus$ be an arbitrary $\mathit{OneNatBin}$-algebra with carrier a; so $e\colon\ \mathit{1} \to a$, $f\colon\ \mathit{nat} \to a$, $\oplus\colon\ \mathit{II}a \to a$. (To be continued.)

5 *Homomorphism, category of algebras, catamorphism.* Fix a default category. Let F be an endofunctor. Given F-algebras φ and ψ, a morphism f is an F-*homomorphism from* φ *to* ψ, denoted $f\colon\ \varphi \to_F \psi$, if: $\varphi \mathbin{;} f = Ff \mathbin{;} \psi$. Further, $\mathcal{A}lg(F)$ is: the category whose objects are F-algebras, whose morphisms are F-homomorphisms, and whose composition and identities are inherited from the underlying default category; thus $\to_F$ is short for $\to_{\mathcal{A}lg(F)}$. (Actually, $\mathcal{A}lg(F)$, thus defined, is not a category but a pre-category, since the source and target of a morphism are not necessarily unique. An easy fix is to take as the morphisms the triples (φ, f, ψ) such that $f\colon \varphi \to_F \psi$, defining composition and identities in the obvious way. To keep the notation light, we shall however continue to write just the f components.)

An F-algebra α is *initial* in $\mathcal{A}lg(F)$ (or: is an initial F-algebra) if: for each F-algebra φ there exists precisely one homomorphism from α to φ; this one is denoted $(\!|\alpha \to \varphi|\!)_F$, or simply $(\!|\varphi|\!)$ if F and α are understood, and it is called a *catamorphism*. If an initial F-algebra exists, we let μF denote one. (All initial F-algebras are isomorphic.) We shall use α, β to denote initial algebras.

6 *Example: trees continued.* The algebra of trees may be defined by

$$nil \triangledown tip \triangledown join \quad = \quad \mu\, OneNatBin\,,$$

or, equivalently,

$$\begin{aligned} \alpha &= \mu\, OneNatBin \\ nil &= in_{3,0} \,;\, \alpha \\ tip &= in_{3,1} \,;\, \alpha \\ join &= in_{3,2} \,;\, \alpha\,. \end{aligned}$$

Let furthermore $\varphi = \underline{0} \triangledown \underline{1} \triangledown (+)$. Then the statement $size\colon \alpha \to_{OneNatBin} \varphi$ is equivalent to the three equations

$$\begin{aligned} nil \,;\, size &= \underline{0} \\ tip \,;\, size &= \underline{1} \\ join \,;\, size &= I\!I\, size \,;\, (+)\,. \end{aligned}$$

In general such equations may have none, one, or more solutions for $size$. But since α is initial, these equations have precisely one solution: the function that yields the number of tips in the tree, and we can write

$$size \quad = \quad (\!|nil \triangledown tip \triangledown join \to \underline{0} \triangledown \underline{1} \triangledown (+)|\!)\,,$$

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 $nil, tip, join \ := \ \underline{0}, \underline{1}, (+)$, as suggested by the notation for catamorphisms. For example, $size((tip\, x) +\!\!+\ nil) +\!\!+\ (tip\, y)) = ((1 + 0) + 1)$.

7 *Calculation rules for catamorphisms.* We list here some facts that follow from initiality and that we need in the sequel. The existence of a mapping $(\!|\alpha \to \_|\!)$ satisfying rule cata-Charn is just the *definition* of the initiality of α; the other rules follow quite easily. The name 'Charn' is mnemonic for 'Characterisation', and the name 'Self' suggests that $(\!|\varphi|\!)$ itself is a solution for x in the left-hand side of rule Charn. Fix a default category and an endofunctor F. Suppose α is an initial F-algebra, and φ is just an F-algebra. Then, abbreviating $(\!|\alpha \to \_|\!)$ to $(\!|\_|\!)$,

$$x\colon \alpha \to_F \varphi \qquad \equiv \qquad x = (\!|\varphi|\!) \qquad\qquad \text{cata-Charn}$$

$$(\!|\varphi|\!)\colon\ \alpha \to_F \varphi \qquad\qquad \text{cata-S}{\small\text{ELF}}$$

$$x\colon\ \varphi \to_F \psi \quad\Rightarrow\quad (\!|\varphi|\!)\ ;\ x = (\!|\psi|\!) \qquad\qquad \text{cata-F}{\small\text{USION}}$$

Remember that $x\colon\ \varphi \to_F \psi$ just means that x satisfies the equation $\varphi\ ;\ x = Fx\ ;\ \psi$. Fokkinga (1992a) exploits such rules to prove various results from category theory, by calculation rather than diagram chasing.

Many more calculation properties of algebras (and co-algebras), and their use in program development, have been given by Malcolm (1990a), Fokkinga (1992b), and Meijer et al. (1991).

8 *Dialgebra.* Dually to the notion of algebra we might define a *G-co-algebra* to be a morphism $\varphi\colon\ a \to Ga$, for some a called the *carrier of* φ. (The importance of co-algebras for programming is illustrated by Hagino (1987), Malcolm (1990b), and Meijer et al. (1991). We shall hardly consider co-algebras in this paper.)

There is also a generalisation that covers both algebras and co-algebras. A morphism $\varphi\colon\ Fa \to Ga$, for some a, is called an *F, G-dialgebra.* For F, G-dialgebras φ and ψ, we say f is an *F, G-dialgebra homomorphism* from φ to ψ, denoted $f\colon\ \varphi \to_{F,G} \psi$, if: $\varphi\ ;\ Gf = Ff\ ;\ \psi$. (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.) This gives rise to a category $\mathcal{D}iAlg(F, G)$. Its objects are dialgebras, its morphisms are dialgebra homomorphisms, and the composition and identities are inherited from the underlying category. (To be precise, rather than taking dialgebras φ as objects in $\mathcal{D}iAlg(F, G)$, we should take pairs (φ, a), satisfying $\varphi\colon\ Fa \to Ga$, since we must be able to retrieve the carrier a from such an object.) The so-called forgetful functor $U\colon\ \mathcal{D}iAlg(F, G) \to \mathcal{A}$, where $\mathcal{A}$ is the source category of F and G, is defined as follows: $U\varphi$ = the carrier of φ, and $Uf = f$. Thus '$U\varphi$' is a handy abbreviation of 'the carrier of φ'. We shall use the notion of dialgebra mainly in the description of 'transformer' below.

The normal, co- and di- algebras defined above are 'single-sorted' with respect to the default category, since their carrier is just one object in the category. However, the algebras are n-sorted in $\mathcal{A}$ if the default category is an n-fold product category $\mathcal{A}^n$. We will not use and discuss 'many-sortedness' in this paper. The notion of datatype is discussed in a little more detail in Section H.

C. Transformer and Law

9 *Abstracting from syntax.* Conventionally 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, the two terms 'x' and '$x\,\mathrm{join}\,x$' (with variable x of type *tree*) are modeled by morphisms *id* and $id \vartriangle id\ ;\ join$ of type $tree \to tree$. So, an equation for φ is modeled by a pair of terms $(T\varphi, T'\varphi)$, T and T' being mappings of morphisms which we call *'transformers'*. This faces us with the following problem: what properties must we require of an arbitrary mapping T in order that it model 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 $T\varphi$ 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 Id-algebras we disallow the following mappings as transformer.

$$\begin{array}{lcl} T\varphi & = & \textbf{if } \varphi \text{ has carrier } \mathit{nat} \textbf{ then } \mathit{succ} \textbf{ else } \varphi \\ T\varphi & = & \textbf{if } \varphi \text{ is bijective } \textbf{then} \text{ the inverse of } \varphi \textbf{ else } \varphi\,. \end{array}$$

We exclude these mappings not only for intuitive reasons, but also because they do not have the properties we want to hold. A tentative definition that a transformer is a natural transformation in the underlying default category does meet our intuitive wish and enables us to prove several desirable theorems, but it makes some terms unexpressible as a transformer (see Theorem 24). 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 conventionally is done up to now, and what we wish to avoid.

10 *A property observed.* Our solution is to impose a property, saying that homomorphisms are mapped to homomorphisms. 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 17 and 19 are just the formalisation of the argument here.) Suppose $\varphi\colon a \to a$ and $T\varphi\colon Ha \to Ja$.

Following Meertens (1989) we view 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. You can also easily construct boxes for the duplication $id \vartriangle id$, and for the swap $\mathit{exr} \vartriangle \mathit{exl}$. Now imagine a box (term) $T\varphi\colon Ha \to Ja$ built with several copies of a box for $\varphi\colon a \to a$, and assume that for some $f\colon a \to b$ and $\psi\colon b \to b$ the equality $\varphi \mathbin{;} f = f \mathbin{;} \psi$ holds. Suppose you insert on each of the output lines a box for f, thus forming a composite box $T\varphi \mathbin{;} Jf\colon Ha \to Jb$. 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. Since $\varphi \mathbin{;} f = f \mathbin{;} \psi$ you can push the box for f through that for φ while replacing the latter by a box for ψ, and continue shifting the box for f along the lines. In this way, eventually, f is shifted to the input gates. Thus, if $\varphi \mathbin{;} f = f \mathbin{;} \psi$ then you may expect that $T\varphi \mathbin{;} Jf = Hf \mathbin{;} T\psi$.

11 *Generalisation.* Generalising, in the above observation, $\varphi\colon a \to a$ and $\psi\colon b \to b$ to $\varphi\colon Fa \to Ga$ and $\psi\colon Fb \to Gb$, it is reasonable to expect in the same way that $\varphi \mathbin{;} Gf = Ff \mathbin{;} \psi$ implies $T\varphi \mathbin{;} Jf = Hf \mathbin{;} T\psi$. Using dialgebras we can formulate this as:

(a) $\qquad f\colon \varphi \to_{F,G} \psi \quad \Rightarrow \quad f\colon T\varphi \to_{H,J} T\psi\,.$

Notice that this formula makes sense even if not all the entities are in one and the same category. The most general typing is easily found: there are categories $\mathcal{A}, \mathcal{B}, \mathcal{C}$, the functors are typed $F, G\colon \mathcal{A} \to \mathcal{B}$ and $H, J\colon \mathcal{A} \to \mathcal{C}$. It follows that f is in $\mathcal{A}$, φ, ψ are in $\mathcal{B}$, and $T\varphi, T\psi$ are in $\mathcal{C}$. More precisely, if $\varphi\colon Fa \to_{\mathcal{B}} Ga$, then $T\varphi\colon Ha \to_{\mathcal{C}} Ja$; so T preserves the carrier of its argument.

We shall now derive two alternative but equivalent formulations of property (a).

Functoriality. Notice that, apparently, T sends $\mathcal{D}iAlg(F,G)$-objects to $\mathcal{D}iAlg(H,J)$-objects. Actually, if we extend T by *defining* $Tf = f$ for each $\mathcal{D}iAlg(F,G)$-morphism f, then property (a) above is one of the axioms for T to be a functor

(b.0) $\quad T \quad : \quad \mathcal{D}iAlg(F,G) \to \mathcal{D}iAlg(H,J)$.

The other functor axioms are the equations $T\mathit{id} = \mathit{id}$ and $T(f \mathbin{;} g) = Tf \mathbin{;} Tg$; these are trivially valid by defining $Tf = f$. Thus extended, T is a functor indeed. That T is the identity on the morphisms in $\mathcal{D}iAlg(F,G)$ can also be formalised as

(b.1) $\quad U'T \quad = \quad U$,

where U, U' are the appropriate forgetful functors,

$$U\colon \mathcal{D}iAlg(F,G) \to \mathcal{A} \qquad \text{and} \qquad U'\colon \mathcal{D}iAlg(H,J) \to \mathcal{A} .$$

Clearly, a T satisfying (a) can be extended to a T satisfying (b.0) and (b.1), and conversely, a T satisfying (b.0) and (b.1) also satisfies (a).

Naturality. There is still another reading of the typing of T and property (a), namely

$$T\varphi\colon HU\varphi \to JU\varphi \qquad \text{for each } F,G\text{-dialgebra } \varphi$$
$$HUf \mathbin{;} T\psi \;=\; T\varphi \mathbin{;} JUf \qquad \text{for each } f\colon \varphi \to_{F,G} \psi ,$$

where $U\colon \mathcal{D}iAlg(F,G) \to \mathcal{A}$. So, T is a natural transformation in $\mathcal{C}$ from HU to JU,

(c) $\quad T \quad : \quad HU \mathrel{\dot{\to}} JU$.

And each T satisfying (c) also satisfies (a). This was observed by Ross Paterson and Peter de Bruin. The latter also pointed out that transformers are a —slight— generalisation of the *semantic operations* described by Manes (1976). Manes investigates a relation with *syntactic operations*, but doesn't discuss most topics of this chapter.

In the following definition we choose one of the three equivalent ways (a), (b.0,b.1) and (c) to characterise transformers.

12 Definition. (Transformer, Law) *Let $F, G\colon \mathcal{A} \to \mathcal{B}$ and $H, J\colon \mathcal{A} \to \mathcal{C}$ be functors. Then a* transformer *of* type $(F,G) \to (H,J)$ *is: a mapping T from F,G-dialgebras to H,J-dialgebras satisfying*

$$f\colon \varphi \to_{F,G} \psi \qquad \Rightarrow \qquad f\colon T\varphi \to_{H,J} T\psi \qquad \textsc{Transformer}$$

that is,

$$\varphi \mathbin{;} Gf = Ff \mathbin{;} \psi \qquad \Rightarrow \qquad T\varphi \mathbin{;} Jf = Hf \mathbin{;} T\psi$$

for all $\mathcal{A}$*-objects* a, b*, and morphisms* $f\colon a \to_{\mathcal{A}} b$*,* $\varphi\colon Fa \to_{\mathcal{B}} Ga$*, and* $\psi\colon Fb \to_{\mathcal{B}} Gb$*. The pairs* (F, G) *and* (H, J) *are called the* source type *and* target type *respectively.*

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:* $T\varphi = T'\varphi$*. Alternatively we also say* $E\varphi$ holds *or* φ satisfies E*; a more formal notation would be* $\models E\varphi$ *or* $\varphi \models E$*.*

A conditional law *is: a pair* E, E' *of laws, both having the same source type, that is, both being applicable to the same dialgebras. Such a law holds for* φ *if:* $E\varphi$ *implies* $E'\varphi$*. (We shall hardly discuss conditional laws.)*

Often we will take $\mathcal{A} = \mathcal{B} = \mathcal{C}$ in applications of transformers, so that F, G, H, J are endofunctors. 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 by taking $\mathcal{B}$ to be a suitable product category. For example, when $\mathcal{B} = \mathcal{B}' \times \mathcal{B}'$, then the transformer gets as argument a pair from $\mathcal{B}'$. This will occur in Section H.

13 *Use of laws.* If a law is prescribed for an F-algebra φ, then of course the law must have source type (F, Id), that is, $G = \mathit{Id}$. The definition of law and transformer may seem unnecessarily general for this application. However, in composing transformers we need the more general form with $G \neq \mathit{Id}$, even though the entire composite transformer has $G = \mathit{Id}$; see Theorem 19. A dual remark holds for co-algebras. As regards to the target type a similar observation holds; in this case either $H = \mathit{Id}$ or $J = \mathit{Id}$ 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 H.

14 *Example: trees continued.* Consider the law "$x \operatorname{join} y = y \operatorname{join} x$," which we shall formalise later. Here the type of the two terms, viewed as functions of x and y, is $\mathit{II}\,\mathit{tree} \to \mathit{tree}$, where tree is the carrier of the transformed algebra. So the transformers that model these terms have target type $(\mathit{II}, \mathit{Id})$, that is, $J = \mathit{Id}$ (and $H = \mathit{II}$). The law induces an equivalence relation on tree that is a congruence for the algebra, namely the least equivalence relation that contains all pairs $(x \operatorname{join} y,\ y \operatorname{join} x)$ (as indicated by the law) and is closed under the operations of the algebra, meaning that with (x, x') and (y, y') it also contains $(\mathit{tip}\, x,\ \mathit{tip}\, x')$ and $(x \operatorname{join} y,\ x' \operatorname{join} y')$. Imposing the law on the algebra means to identify equivalent elements and to consider the induced quotient algebra.

Now consider the law "$\mathit{size}\, x \bmod 2 = 0$" (also formalised later). Here the type of the two terms, viewed as functions of x, is $\mathit{tree} \to \mathit{nat}$, so, since tree is the carrier of the transformed algebra, in this case the transformers have target type $(\mathit{Id}, \underline{\mathit{nat}})$, that is, $H = \mathit{Id}$ (and $J = \underline{\mathit{nat}}$). 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 $G = \mathit{Id} = J$: applicable to algebras and meant to identify elements of the carrier. We are in fact primarily interested in the rôle of the Transformer property, 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.

15 *Example: trees continued.* By making *nil* neutral for *join* (that is, making *nil* the identity for *join*) it behaves properly as 'empty': joining *nil* to a structure yields the same structure again.

By further imposing associativity of *join* the trees become effectively lists or sequences, known as *join lists*: when $x \,\mathrm{join}\, (y \,\mathrm{join}\, z) = (x \,\mathrm{join}\, y) \,\mathrm{join}\, z$, the parentheses may be omitted, and that structure can be denoted by $x \,\mathrm{join}\, y \,\mathrm{join}\, z$, the usual notation for a list.

Bags result by imposing commutativity of *join* as well: when $x \,\mathrm{join}\, y = y \,\mathrm{join}\, x$, the order in which the elements are joined to a structure is insignificant.

Finally, *sets* are obtained if *join* is made absorptive (idempotent) too: when $x \mathrm{join} x = x$, the multiplicity of the elements in a structure is insignificant, as for sets.

Meertens (1986) attributes to H.J. Boom the above observation that trees, lists, bags, and sets are thus related. 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 $\varphi = e \triangledown f \triangledown \oplus$: $1 + nat + I\!I a \to a$ be an arbitrary *OneNatBin* -algebra. Observe that the constituent operations of φ can be expressed as follows.

$$\begin{array}{lcllcl} e & = & in_{3,0} \,;\, \varphi & : & 1 \to a \\ f & = & in_{3,1} \,;\, \varphi & : & nat \to a \\ \oplus & = & in_{3,2} \,;\, \varphi & : & I\!I a \to a\,. \end{array}$$

So when we say $T\varphi = \ldots e \ldots \oplus \ldots$, we actually mean the right-hand side that is obtained by substituting the above definitions for $e, f,$ and $\oplus$. We discuss the simplest laws first.

Absorptivity. In order to express $x \oplus x = x$ for all x in a , take

$$T\varphi = split \,;\, \oplus \qquad \text{and} \qquad T'\varphi = id\,,$$

where $split = id \vartriangle id$. Here and in the following examples, Theorems 19, 20, and 21 imply the validity of the TRANSFORMER property for both T and T' on account of the way they are composed out of basic transformers. But it may be instructive to verify the property at least once explicitly. We do it here for T , the verification for T' is trivial.

Consider two arbitrary *OneNatBin* -algebras $\varphi = d \triangledown g \triangledown \oplus$ and $\psi = e \triangledown h \triangledown \otimes$. Suppose f is a homomorphism from φ to ψ ,

$$f \quad : \quad \psi \to_{OneNatBin} \psi$$

that is,

$$\begin{array}{lcl} d \,;\, f & = & e \\ g \,;\, f & = & h \\ \oplus \,;\, f & = & I\!I f \,;\, \otimes\,. \end{array}$$

Then f is a homomorphism from $T\varphi$ to $T\psi$:

$$\begin{array}{ll} & f\colon\ T\varphi \rightarrow_{Id,Id} T\psi \\ \equiv & \quad \text{definition } T \text{, definition } \rightarrow \\ & split \mathbin{;} \oplus \mathbin{;} f = f \mathbin{;} split \mathbin{;} \otimes \\ \equiv & \quad \text{lhs: assumption on } f \text{, rhs: naturality } split\colon\ Id \dot{\rightarrow} I\!I \\ & split \mathbin{;} I\!I f \mathbin{;} \otimes = split \mathbin{;} I\!I f \mathbin{;} \otimes \\ \equiv & \quad \text{equality} \\ & true. \end{array}$$

Commutativity. To express $x \oplus y = y \oplus x$ for all x, y in a, take

$$T\varphi \;=\; \oplus \qquad \text{and} \qquad T'\varphi \;=\; swap \mathbin{;} \oplus ,$$

where $swap = exr \vartriangle exl$.

Neutrality. To express $e \oplus x = x$ for all x in a, take

$$T\varphi \;=\; (! \mathbin{;} e) \vartriangle id \mathbin{;} \oplus \qquad \text{and} \qquad T'\varphi \;=\; id .$$

Here $!\colon\ a \rightarrow 1$ is the unique morphism into the unit type 1.

Associativity. To express $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ for all x, y, z in a, take

$$T\varphi \;=\; \oplus \times id \;\; ; \;\; \oplus \qquad \text{and} \qquad T'\varphi \;=\; assoc \;\; ; \;\; id \times \oplus \;\; ; \;\; \oplus$$

where

$$assoc \;=\; (exl \mathbin{;} exl) \vartriangle ((exl \mathbin{;} exr) \vartriangle exr) :\ \ (X \times Y) \times Z \dot{\rightarrow} X \times (Y \times Z)$$

Here functors X, Y, Z stand for $Ex_{3,0}, Ex_{3,1}, Ex_{3,2}$.

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, $size$ is just $(\!|\varphi \rightarrow zero \triangledown one \triangledown add|\!)$, as we have shown earlier. However, T should be applicable to every 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 $size$: "$\varphi \mathbin{;} \psi = F\psi \mathbin{;} zero \triangledown one \triangledown add$". This is discussed in detail in Section H and gives an F, G-bialgebra (φ, ψ) for some G. Then form, for arbitrary F, G-bialgebra (φ, ψ), the law E suggested by

$$E(\varphi, \psi) \quad = \quad \text{“}\ \varphi \mathbin{;} \psi \mathbin{;} mod2 = zero\ \text{”}\ .$$

Transformers are applicable to bialgebras indeed, by a suitable instantiation of $\mathcal{A}$, $\mathcal{B}$, and the functors in the definition of transformer.

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

16 Fact. *The morphism mappings given at the beginning of the section are not transformers; the* TRANSFORMER *property is not valid for them.*

D. Expressiveness of transformers and laws

We shall see in this section that the TRANSFORMER property for a mapping of type $(F, G) \to (H, J)$ follows from the Theorems For Free theorem (provided it is applicable to the mapping). Further, the usual syntactic ways to compose terms are also applicable to transformers: they are closed under composition and substitution, and the identity mapping and each functor and constant mapping is a transformer. Thus transformers are at least as expressive as syntactic terms. Also, natural transformations of a higher type are transformers, but not conversely. And, finally, laws are closed under conjunction.

* * *

17 Theorem. *Let* T *be a 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 (1989) is applicable to* T *. Then* T *is a transformer of type* $(F, G) \to (H, J)$ *.*

Proof. We use the notation of Wadler (1989) except for our choice of identifiers and the order of composition:

Each function f denotes a relation, namely $(x, y){\in}f \;\equiv\; f(x){=}y$. 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:

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

Here, $a{\Leftrightarrow}b$ is the type of relations containing pairs (x, y) with $x \in a$ and $y \in b$. All morphisms (functions) are required to be total, so that $f \subseteq g$ equivales $f = g$.

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

$$\begin{array}{ll} & T\colon\; \forall\alpha :: (F\alpha \to G\alpha) \to (H\alpha \to J\alpha) \\ \Rightarrow & \qquad \text{Theorems for Free — applicability assumed} \\ & (T, T) \in \forall r :: (Fr \to Gr) \to (Hr \to Jr) \\ \equiv & \qquad \text{definition } \forall \\ & \forall \text{reln } R\colon\; a \Leftrightarrow b. \quad (T_a, T_b) \in (FR \to GR) \to (HR \to JR) \\ \equiv & \qquad \text{definition } \to \text{ (second formulation)} \\ & \forall \text{reln } R\colon\; a \Leftrightarrow b. \forall\varphi, \psi.\; (\varphi, \psi) \in (FR \to GR) \;\Rightarrow\; (T_a\varphi, T_b\psi) \in (HR \to JR) \\ \equiv & \qquad \text{definition } \to \text{ (first alternative) at both sides} \\ & \forall \text{reln } R\colon\; a \Leftrightarrow b. \forall\varphi, \psi. \quad FR \,;\, \psi \subseteq \varphi \,;\, GR \;\Rightarrow\; HR \,;\, T_b\psi \subseteq T_a\varphi \,;\, JR \\ \Rightarrow & \qquad \text{taking } R\colon\; a \Leftrightarrow b \text{ to be a function } f\colon\; a \to b \\ & \forall \text{fctn } f\colon\; a \to b. \forall\varphi, \psi. \quad Ff \,;\, \psi = \varphi \,;\, Gf \;\Rightarrow\; Hf \,;\, T_b\psi = T_a\varphi \,;\, Jf \end{array}$$

which is the required TRANSFORMER property. (To be *very* precise, it is not T , but T' defined by $T'\varphi = T_{carrier\,\varphi}\varphi$, that is a transformer.) □

One condition on T for the applicability of the Theorems For Free theorem is that T is lambda-definable (and there are some more conditions on the category). In the definition of transformer each morphism mapping is allowed, even those that are not lambda-definable. Theorems For Free suggests that TRANSFORMER is a crucial property (and provides an alternative rationale for requiring this property to hold for transformers). Moreover, working in a 'functional' categorical setting, it seems that Theorems For Free suggests no stronger property for transformers.

18 *Composite transformers.* Here follow some theorems showing how transformers may be composed to form transformers again.

19 Theorem. *The following equations define transformers of the type indicated, provided that T, T' are transformers of type $(F,G) \to (H,J)$ and $(F',G') \to (H',J')$ respectively, and that the well-formedness conditions at the right hold.*

definition	typing	condition
$I\varphi = \varphi$	$I : (F,G) \to (F,G)$	
$\underline{f}\varphi = f$	$\underline{f} : (F,G) \to (\underline{\mathrm{src} f}, \underline{\mathrm{tgt} f})$	
$\varepsilon'\varphi = \varepsilon_{U\varphi}$	$\varepsilon' : (F,G) \to (H,J)$	$\varepsilon\colon\ H \dot{\to} J$
$(T;T')\varphi = T\varphi \mathbin{;} T'\varphi$	$(T;T') : (F,G) \to (H,J')$	$\begin{cases}(F,G) = (F',G') \\ J = H'\end{cases}$
$(T \circ T')\varphi = T(T'\varphi)$	$(T \circ T') : (F',G') \to (H,J)$	$(H',J') = (F,G)$
$\llparenthesis\_\rrparenthesis_F\, \varphi = \llparenthesis\varphi\rrparenthesis_F$	$\llparenthesis\_\rrparenthesis : (F, \mathit{Id}) \to (\underline{U\mu F}, \mathit{Id})$	μF exists.

Proof. The correct typing is immediate for all these transformers. As regards the TRANSFORMER property for $\llparenthesis\_\rrparenthesis$ we argue as follows. Let $a = U\mu F$. Then

$$
\begin{array}{ll}
 & \underline{a}f \mathbin{;} \llparenthesis\_\rrparenthesis\varphi = \llparenthesis\_\rrparenthesis\psi \mathbin{;} \mathit{Id}f \\
\equiv & \qquad \text{definition } \llparenthesis\_\rrparenthesis,\ \underline{a},\ \text{and } \mathit{Id} \\
 & \llparenthesis\varphi\rrparenthesis = \llparenthesis\psi\rrparenthesis \mathbin{;} f \\
\Leftarrow & \qquad \text{cata-FUSION} \\
 & \psi \mathbin{;} f = Ff \mathbin{;} \varphi.
\end{array}
$$

As regards the TRANSFORMER property of the composite $T \circ T'$ we argue:

$$
\begin{array}{ll}
 & T(T'\varphi) \mathbin{;} Jf = Hf \mathbin{;} T(T'\psi) \\
\Leftarrow & \qquad \text{TRANSFORMER } T\text{, noting that } (F,G) = (H',J') \\
 & T'\varphi \mathbin{;} J'f = H'f \mathbin{;} T'\psi \\
\Leftarrow & \qquad \text{TRANSFORMER } T' \\
 & \varphi \mathbin{;} G'f = F'f \mathbin{;} \psi.
\end{array}
$$

Rephrased with the $\to_{F,G}$ notation, this calculation is but a special instance of the proof that the composition of functors is a functor again.
The other parts are proved similarly to $T \circ T'$. Actually, that $\underline{f}$ is a transformer follows also from the fact that ε' is a transformer, since $f\colon\ a \to b$ implies $\underline{f}\colon\ \underline{a} \dot{\to} \underline{b}$. □

20 Theorem. *Let T be a transformer of type $(F, G) \to (H, J)$, and K a functor into the source category of F, G, H, J. Then T is also a transformer of type $(FK, GK) \to (HK, JK)$.*

Proof. The typing is clearly correct. As regards the TRANSFORMER property we argue:

$$\begin{array}{ll} & f\colon\ T\varphi \to_{HK,JK} T\psi \\ \equiv & \qquad \text{unfold, fold} \\ & Kf\colon\ T\varphi \to_{H,J} T\psi \\ \Leftarrow & \qquad \text{TRANSFORMER for } T \text{ of type } (F,G) \to (H,J) \\ & Kf\colon\ \varphi \to_{F,G} \psi \\ \equiv & \qquad \text{unfold, fold} \\ & f\colon\ \varphi \to_{FK,GK} \psi \end{array}$$

as required. □

The next theorem shows that each functor is a transformer. Remember that *Exl* and *Exr* denote the extraction (projection) functors from a product category to the component categories respectively.

21 Theorem. *Let $K\colon \mathcal{B} \to \mathcal{C}$ be a functor. Put $X, Y = \mathit{Exl}, \mathit{Exr}$, both being functors of type $\mathcal{B} \times \mathcal{B} \to \mathcal{B}$. Then K is a transformer of type $(X, Y) \to (KX, KY)$.*

Proof. The typing requirement for K is met: taking $\mathcal{A} = \mathcal{B} \times \mathcal{B}$,

$$\begin{array}{ll} & \forall a \text{ in } \mathcal{A},\ \varphi\colon\ Xa \to_{\mathcal{B}} Ya ::\quad K\varphi\colon\ KXa \to_{\mathcal{C}} KYa \\ \equiv & \qquad \text{definition } \mathcal{A} \text{ and } X, Y \\ & \forall b, c \text{ in } \mathcal{B},\ \varphi\colon\ b \to_{\mathcal{B}} c ::\quad K\varphi\colon\ Kb \to_{\mathcal{C}} Kc \\ \equiv & \qquad \text{functoriality of } K \\ & \mathit{true}. \end{array}$$

To check the TRANSFORMER property, we argue:

$$\begin{array}{ll} & f\colon\ K\varphi \to_{KX,KY} K\psi \\ \Leftarrow & \qquad \text{general theorem (even for arbitrary } X, Y\text{)} \\ & f\colon\ \varphi \to_{X,Y} \psi. \end{array}$$

It may be instructive to spell out this implication. Observe that a morphism f in $\mathcal{B} \times \mathcal{B}$ has the form $f = (g, h)$ for some morphisms g, h in $\mathcal{B}$. The TRANSFORMER property thus reads

$$(g, h)\colon\ \varphi \to_{\mathit{Exl},\mathit{Exr}} \psi \quad \Rightarrow \quad (g, h)\colon\ K\varphi \to_{K\,\mathit{Exl},\ K\,\mathit{Exr}} K\psi$$

that is,

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

Indeed, this is valid for each functor K. □

From all these theorems we conclude that for all conventional laws there is no need to check TRANSFORMER explicitly: the transformers of such laws are built entirely by the compositions of the theorems. In particular this holds for morphisms and natural transformations that arise from products and sums.

22 *Another naturality property of transformers.* Before we realised that transformers are natural transformations as explained in paragraph 11 we were looking for naturality properties in the way reported here. As a motivation, notice that a transformer T maps morphisms of type $Fa \to Ga$ into morphisms of type $Ha \to Ja$. In a sense, transformers are natural transformations from 'functor' $F \to G$ to 'functor' $H \to J$. Let us first make precise what we mean by 'functor' $F \to G$, taking care of the arising contravariance.

We write the Hom-functor with an infix symbol $\_ \to \_$:

$$\_ \to \_ \quad : \quad \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{S}et\,.$$

Recall its definition: For objects a, b, c, d and morphisms $f\colon a \to b$ and $g\colon c \to d$,

$$\begin{array}{lcll} a \to b & = & \{x| \quad x\colon a \to_{\mathcal{C}} b\} & \\ f \to g & = & \lambda(x :: \quad f \,;\, x \,;\, g) & : \quad (b \to c) \to (a \to d)\,. \end{array}$$

The interchange of a and b in the type of $(f \to g)$ means that $\_ \to \_$ is contravariant in its first argument, indicated by ^{op} in the typing. Notice that $x\colon a \to_{\mathcal{C}} b$ equivales $x \in a \to b$. For readability put $X, Y = \mathit{Exl}, \mathit{Exr}$. Recall also the convention that $(F \dagger G)x = Fx \dagger Gx$, which we shall use with $\to$ for $\dagger$. With these conventions we have two equally typed functors:

$$\begin{array}{lcl} (FX \to GY) & : & \mathcal{A}^{op} \times \mathcal{B} \to \mathcal{S}et \\ (HX \to JY) & : & \mathcal{A}^{op} \times \mathcal{B} \to \mathcal{S}et \end{array}$$

whenever

$$\begin{array}{lcll} F & : & \mathcal{A} \to \mathcal{C} & \text{(hence } F\colon \mathcal{A}^{op} \to \mathcal{C}^{op}\text{)} \\ G & : & \mathcal{B} \to \mathcal{C} & \\ H & : & \mathcal{A} \to \mathcal{D} & \text{(hence } H\colon \mathcal{A}^{op} \to \mathcal{D}^{op}\text{)} \\ J & : & \mathcal{B} \to \mathcal{D}\,. & \end{array}$$

Consider now a natural transformation $T\colon (FX \to GY) \mathrel{\dot\to} (HX \to JY)$, where F, G, H, J are typed as above. Working out in detail what naturality means, we find

$$\begin{array}{ll} & T\colon (FX \to GY) \mathrel{\dot\to} (HX \to JY) \\ \equiv & \qquad \text{naturality: for all } (f,g)\colon (a,b) \to_{\mathcal{A}^{op}\times\mathcal{B}} (a',b') \text{ (so } f\colon a' \to_{\mathcal{A}} a\text{)} \\ & (Ff \to Gg) \,;\, T_{a'b'} = T_{ab} \,;\, (Hf \to Jg) \\ \equiv & \qquad \text{extensionality in } \mathcal{S}et\text{: for all } \varphi \in (Fa \to Gb) \\ & ((Ff \to Gg) \,;\, T_{a'b'})\varphi = (T_{ab} \,;\, (Hf \to Jg))\varphi \\ \equiv & \qquad \text{application, composition, hom-functor} \\ & T_{a'b'}(Ff \,;\, \varphi \,;\, Gg) = Hf \,;\, T_{ab}\varphi \,;\, Jg \\ \equiv & \qquad \text{for } \Rightarrow\text{: instantiation with } a, a', f := a, a, id\text{, and } b, b', g := b, b, id\,, \\ & \qquad \text{for } \Leftarrow\text{: use (ntrf0) with } a, \varphi := a', (Ff, \varphi)\text{, followed by (ntrf1)} \end{array}$$

(ntrf0) $\quad T_{ab'}(\varphi \mathbin{;} Gg) = T_{ab}\varphi \mathbin{;} Jg \qquad \wedge$

(ntrf1) $\quad T_{a'b}(Ff \mathbin{;} \varphi) = Hf \mathbin{;} T_{ab}\varphi.$

That is, natural transformation T satisfies a two-sided fusion law. (An adjunction between F and J is nothing but such a natural transformation that has an inverse, in which case necessarily $G = H = Id$.) Notice that the subscript (a, b) in $T_{ab}\varphi$ may be expressed in terms of φ by using the forgetful functor $U\colon \mathcal{D}iAlg(FX, GY) \to \mathcal{A} \times \mathcal{B}$, namely $(a, b) = U\varphi$.

With F, G, H, J as above, consider now a mapping T' of type $(FX, GY) \to (HX, JY)$. Working out in detail what the TRANSFORMER property means, we find:

$$\begin{array}{ll}
 & T' \text{ is a transformer} \\
\equiv & \quad \text{definition TRANSFORMER:} \\
 & \quad \text{For all } (FX, GY)\text{-dialgebras } \varphi \text{ and } \psi, \\
 & \quad \text{and all } (f, g) \text{ of the appropriate type} \\
 & (f, g)\colon \varphi \to_{FX,GY} \psi \qquad \Rightarrow \qquad (f, g)\colon T'\varphi \to_{HX,JY} T'\psi \\
\equiv & \quad \text{definition } \to, \text{ definition } X, Y
\end{array}$$

(trafo) $\quad \varphi \mathbin{;} Gg = Ff \mathbin{;} \psi \qquad \Rightarrow \qquad T'\varphi \mathbin{;} Jg = Hf \mathbin{;} T'\psi.$

We are now prepared to relate the transformer property to naturality in the underlying category.

23 Theorem. (Meertens) *Let functors F, G, H, J, and the forgetful functor U be typed as above in paragraph 22. Suppose T is a transformation (not required to be natural) of type $(FX \to GY) \to (HX \to JY)$. Define mapping T' by $T'\varphi = T_{U\varphi}\varphi$. Then T' maps (FX, GY)-dialgebras to (HX, JY)-dialgebras with preservation of the carrier, and:*

$$\begin{array}{ll}
 & T \text{ is natural } (\ \text{ of type } (FX \to GY) \mathrel{\dot{\to}} (HX \to GY)\ ) \\
\equiv & \\
 & T' \text{ is a transformer of type } (FX, GY) \to (HX, GY).
\end{array}$$

Proof. The claim about the type of T' is immediate; as a consequence the equivalence makes sense. For part $\Rightarrow$ of the equivalence, we establish the required TRANSFORMER property (trafo) as follows. Let $\varphi\colon Fa \to_{\mathcal{C}} Gb = (FX \to GY)(a, b)$ be arbitrary, and $\psi\colon Fa' \to_{\mathcal{C}} Gb'$, and $(f, g)\colon (a', b) \to_{\mathcal{A}\times\mathcal{B}} (a, b')$. Then:

$$\begin{array}{ll}
 & T'\varphi \mathbin{;} Jg = Hf \mathbin{;} T'\psi \\
\equiv & \quad \text{definition } T' \\
 & T_{ab}\varphi \mathbin{;} Jg = Hf \mathbin{;} T_{a'b'}\psi \\
\equiv & \quad \text{in lhs naturality } T: \text{(ntrf0)}, \\
 & \quad \text{in rhs naturality } T: \text{(ntrf1) with } a, b, a', \varphi := a', a, b', \psi \\
 & T_{ab'}(\varphi \mathbin{;} Gg) = T_{ab'}(Ff \mathbin{;} \psi) \\
\Leftarrow & \quad \text{Leibniz} \\
 & \varphi \mathbin{;} Gg = Ff \mathbin{;} \psi.
\end{array}$$

For part $\Leftarrow$ of the equivalence, we firstly establish the required fusion law (ntrf0) as follows:

$$
\begin{array}{ll}
 & T_{ab'}(\varphi \mathbin{;} Gg) = T_{ab}\varphi \mathbin{;} Jg \\
\equiv & \qquad \text{identity, definition } T' \\
 & H\mathit{id} \mathbin{;} T'(\varphi \mathbin{;} Gg) = T'\varphi \mathbin{;} Jg \\
\Leftarrow & \qquad \text{property (trafo) for } T'\text{, with } f,\psi := \mathit{id},(\varphi \mathbin{;} Gg) \\
 & F\mathit{id} \mathbin{;} \varphi \mathbin{;} Gg = \varphi \mathbin{;} Gg \\
\equiv & \qquad \text{identity} \\
 & \mathit{true}.
\end{array}
$$

Secondly, fusion law (ntrf1) may be established similarly: using (trafo) with $g,\psi := \mathit{id},(Ff \mathbin{;} \varphi)$. □

The above theorem was only formulated and proved, after we had found a direct proof of the following corollary.

24 Corollary. *Let F,G,H,J be typed as above in paragraph 22, but with the additional constraint that $\mathcal{A}=\mathcal{B}$. Suppose $T\colon (FX \to GY) \dot{\to} (HX \to JY)$. Define mapping T' by $T'\varphi = T_{U\varphi,U\varphi}\varphi$, where $U\colon \mathcal{D}iAlg(F,G) \to \mathcal{A}$. Then T' is a transformer of type $(F,G) \to (H,J)$. The converse is not true, that is, there exist transformers that cannot be written this way.*

Proof. A direct proof is easy: just adapt part $(\Rightarrow)$ of the proof of Theorem 23. An indirect proof runs as follows. Let Δ be the doubling functor: $\Delta x = (x,x)$. Then:

$$
\begin{array}{ll}
 & T' \text{ is transformer of type } (F,G) \to (H,J) \\
\equiv & \qquad \text{property } X\Delta = \mathit{Id}_{\mathcal{A}} \text{ and } Y\Delta = \mathit{Id}_{\mathcal{B}}\text{ , } \mathcal{A}=\mathcal{B} \\
 & T' \text{ is transformer of type } (FX\Delta, GY\Delta) \to (HX\Delta, JY\Delta) \\
\Leftarrow & \qquad \text{Theorem 20 with } K := \Delta \\
 & T' \text{ is transformer of type } (FX, GY) \to (HX, JY) \\
\Leftarrow & \qquad \text{Theorem 23} \\
 & T\colon (FX \to GY) \dot{\to} (HX \to GY).
\end{array}
$$

The constraint $\mathcal{A}=\mathcal{B}$ is necessary for the phrase 'transformer of type $(F,G) \to (H,J)$ ' to make sense.
To show that the converse is not true, consider arbitrary $\eta\colon H \dot{\to} J$. Then by Theorem 19 $\underline{\eta}$ is a transformer of type $(F,G) \to (H,J)$. It is not a natural transformation of type $(FX \to GY) \dot{\to} (HX \to JY)$ since the typing is not correct; this is also apparent from the two-sided fusion law that now simplifies to

$$\eta \quad = \quad Hf \mathbin{;} \eta \mathbin{;} Jg$$

which should hold for each a,b,c,d and $f\colon a \to b$ and $g\colon c \to d$ — clearly impossible in general. For a counterexample, take $\eta = \mathit{id}\colon \mathit{Id} \dot{\to} \mathit{Id}$. □

25 *Conjunction.* If E_0 and E_1 are two laws with the same source type, then by 'a conjunction' of E_0 and E_1 we mean a law E such that for all φ: $E\varphi \equiv E_0\varphi \wedge E_1\varphi$. We shall show that there are two ways of representing the conjunction of laws. The two ways yield laws of different target type.

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

$$(T_0 \triangledown T_1)\varphi \quad = \quad T_0\varphi \triangledown T_1\varphi \quad : \quad H_0a + H_1a \to Ja$$

for any a and φ: $Fa \to Ga$. It follows that $T_0 \triangledown T_1$ is a mapping of type $(F, G) \to (H_0{+}H_1, J)$. For S_i: $(F, G) \to (H, J_i)$ the composite $S_0 \vartriangle S_1$ is defined similarly, and we have

$$S_0 \vartriangle S_1 \quad : \quad (F, G) \to (H, J_0{\times}J_1)$$

Of course we need to assume that the category has sums or products, respectively.

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

$$(T_0 \triangledown T_1)\varphi = (T_0' \triangledown T_1')\varphi \qquad \equiv \qquad T_0\varphi = T_0'\varphi \ \wedge\ T_1\varphi = T_1'\varphi\,.$$

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

$$(S_0 \vartriangle S_1)\varphi = (S_0' \vartriangle S_1')\varphi \qquad \equiv \qquad S_0\varphi = S_0'\varphi \ \wedge\ S_1\varphi = S_1'\varphi\,.$$

Proof. The equivalences follow from the properties for product and sum. For the TRANSFORMER property of $T_0 \triangledown T_1$, let $\varphi \mathbin{;} Gf = Ff \mathbin{;} \psi$. Then

$$\begin{array}{ll}
 & (T_0 \triangledown T_1)\varphi \mathbin{;} Jf \\
= & \\
 & (T_0\varphi \mathbin{;} Jf) \triangledown (T_1\varphi \mathbin{;} Jf) \\
= & \qquad \text{TRANSFORMER } T_0,\ T_1 \\
 & (H_0f \mathbin{;} T_0\psi) \triangledown (H_1f \mathbin{;} T_1\psi) \\
= & \\
 & (H_0 + H_1)f \mathbin{;} (T_0 \triangledown T_1)\psi.
\end{array}$$

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

So, if two laws $E_0 = (T_0, T_0')$ and $E_1 = (T_1, T_1')$ have typing T_i, T_i': $(F, Id) \to (H_i, Id)$, and are used to "identify elements in the carriers" of F-algebras as explained in paragraphs 13 and 14, then $E = (T_0 \triangledown T_1, T_0' \triangledown T_1')$ is a conjunction of such a type that it may be used for the same purpose as E_0 and E_1. The $\vartriangle$-form of the conjunction of laws is to be used if the laws are used to "leave out elements from the carriers" as explained in paragraph 14.

Of course, there are also arbitrary infinite conjunctions of laws, provided the category has arbitrary infinite sums or products.

E. One half of a Birkhoff characterisation

27 *Birkhoff characterisation.* Let $\mathcal{C}$ be the default category. Let F and H be endofunctors and $E = (T, T')$ be a law of type $(F, Id) \to (H, Id)$, fixed throughout this section. We define $\mathcal{A}lg(F, E)$ as the full subcategory of $\mathcal{A}lg(F)$ 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,

$$\mathcal{A} = \mathcal{A}lg(F, E) \text{ for some law } E$$

if and only if

$$\mathcal{A} \text{ is closed under subalgebras, homomorphic images, and products.}$$

We shall give one half of such an equivalence (the easy half): some closure properties of $\mathcal{A}lg(F, E)$. (I've been unable to prove the converse.) Some care is needed in defining subalgebras and homomorphic images since we wish to work in an arbitrary category, and not just in $\mathcal{S}et$ where several properties hold that are not valid in, say, CPO. The notions of subalgebra and homomorphic image that we define are dual to each other.

28 *Subalgebra.* Given F-algebras φ and ψ, we say φ is a *subalgebra* of ψ if: there exists an $f\colon \varphi \to_F \psi$ which is monic in $\mathcal{C}$. A subcategory $\mathcal{A}$ of $\mathcal{A}lg(F)$ 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 another, related, property. $\mathcal{A}$ is *closed under incoming monos* if: f is a morphism in $\mathcal{A}$ whenever $f\colon \varphi \to_F \psi$ is monic in $\mathcal{C}$ and ψ is in $\mathcal{A}$. For a full subcategory of $\mathcal{A}lg(F)$, closure under subalgebras equivales closure under incoming monos.

29 Theorem. *$\mathcal{A}lg(F, E)$ is closed under subalgebras (i.e., under incoming monos).*

Proof. Suppose $f\colon \varphi \to_F \psi$ is monic, and ψ is in $\mathcal{A}lg(F, E)$. We show that $E\varphi$ holds.

$$\begin{array}{cl} & T\varphi = T'\varphi \\ \Leftarrow & \qquad f \text{ monic} \\ & T\varphi \mathbin{;} f = T'\varphi \mathbin{;} f \\ \equiv & \qquad \text{TRANSFORMER (condition `} \varphi \mathbin{;} f = Ff \mathbin{;} \psi \text{' is satisfied) at both sides} \\ & Hf \mathbin{;} T\psi = Hf \mathbin{;} T'\psi \\ \Leftarrow & \qquad \text{Leibniz, and } \psi \text{ in } \mathcal{A}lg(F, E) \\ & \mathit{true}. \end{array}$$

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

30 *Homomorphic images.* Consider $f\colon \varphi \to_F \psi$. Working in $\mathcal{S}et$ the homomorphic image of φ under f is the algebra ψ restricted to the range of function f. A generalisation to arbitrary categories is problematic, since categorically there are no points available. Working with *varieties*, as Lehmann (1978; 1980) does, the corresponding closure property says that with φ also ψ is in the class. That is certainly not true for $\mathcal{A}lg(F, E)$, as we shall argue after the theorem. Our way out is to consider epic homomorphisms.

We define: A subcategory $\mathcal{A}$ of $\mathcal{A}lg(F)$ is *closed under homomorphic images* if: ψ is in $\mathcal{A}$ whenever there exists an $f\colon \varphi \to_F \psi$ which is epic in $\mathcal{C}$ and φ is in $\mathcal{A}$. And, $\mathcal{A}$ is *closed under outgoing epis* if: f is a morphism in $\mathcal{A}$ whenever $f\colon \varphi \to_F \psi$ is epic and φ is in $\mathcal{C}$. For a full subcategory of $\mathcal{A}lg(F)$, closure under homomorphic images equivales closure under outgoing epis.

31 Theorem. *Suppose H (from the type of E) preserves epis. Then $\mathcal{A}lg(F,E)$ is closed under homomorphic images (that is, under outgoing epis).*

Proof. Let $f\colon \varphi \to_F \psi$ be epic, with φ in $\mathcal{A}lg(F,E)$.

$$
\begin{array}{ll}
 & T\psi = T'\psi \\
\equiv & \qquad Hf \text{ is epic} \\
 & Hf \,;\, T\psi = Hf \,;\, T'\psi \\
\equiv & \qquad \text{TRANSFORMER (condition `} f\colon \varphi \to_F \psi \text{' is satisfied) at both sides} \\
 & T\varphi \,;\, f = T'\varphi \,;\, f \\
\equiv & \qquad \text{assumption: } \varphi \text{ in } \mathcal{A}lg(F,E) \\
 & \mathit{true}.
\end{array}
$$

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

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

32 Lemma. *Let φ in $\mathcal{A}lg(F,E)$, and $f\colon \varphi \to_F \psi$. Then $E\psi$ holds "on the range of f", that is, $Hf \,;\, T\psi = Hf \,;\, T'\psi$.*

The requirement that a functor (like H in the theorem) preserves epis, is a mild one. In $\mathcal{S}et$ all polynomial functors preserve epis. Lehmann (1980) argues that preservation of epis is an important property.

33 *Homomorphisms do not preserve laws.* It is now clear 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 good reason to work with the variety $V_F(E\mu F)$ instead of $\mathcal{A}lg(F,E)$, see Definition 38 and Theorem 39.) For example, imagine in $\mathcal{S}et$ the algebra $\mathit{zero} \triangledown \mathit{add}\colon 1 + \mathit{II}\,\mathit{nat} \to \mathit{nat}$ of finite naturals, where zero is a neutral element for add. Form another algebra by adjoining a fictitious element ω to nat, and extend operation add as follows: for any natural x, $\mathit{add}'(x,\omega) = \mathit{add}'(\omega,x) = x$ and $\mathit{add}'(\omega,\omega) = \omega$. 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'.

34 *Product.* For F-algebras φ, ψ (with carriers a and b say), and F-homomorphisms f, g with common source in $\mathcal{A}lg(F)$ we define

$$
\begin{array}{lcll}
\varphi \times_F \psi & = & (F\mathit{exl} \,;\, \varphi) \vartriangle (F\mathit{exr} \,;\, \psi) & : \quad F(a \times b) \to (a \times b) \\
f \vartriangle_F g & = & f \vartriangle g & \\
\mathit{exl}_F, \mathit{exr}_F & = & \mathit{exl}, \mathit{exr}\,. &
\end{array}
$$

It is then readily verified that $\times_F$, $\vartriangle_F$, exl_F, exr_F form a categorical product in $\mathcal{A}lg(F)$, and we omit the subscript F in these operations since no confusion can result. In particular, exl: $\varphi \times \psi \rightarrow_F \varphi$, that is,

$$(*) \qquad \varphi \times \psi \,;\, exl \quad = \quad F\,exl \,;\, \varphi\,,$$

and similarly for exr. The generalisation to arbitrary products is straightforward; the proviso being that the default category has arbitrary products.

35 Theorem. *$\mathcal{A}lg(F,E)$ is closed under products.*

Proof. We consider only binary products.

$$\begin{array}{ll}
& T(\varphi \times \psi) = T'(\varphi \times \psi) \\
\equiv & \qquad \text{the projections are jointly monic in } \mathcal{C} \\
& T(\varphi \times \psi) \,;\, exl = T'(\varphi \times \psi) \,;\, exl \qquad \text{and} \\
& T(\varphi \times \psi) \,;\, exr = T'(\varphi \times \psi) \,;\, exr \\
\equiv & \qquad \text{Transformer at both sides (condition is satisfied: see } (*) \text{ above)} \\
& H\,exl \,;\, T\varphi = H\,exl \,;\, T'\varphi \qquad \text{and} \\
& H\,exr \,;\, T\psi = H\,exr \,;\, T'\psi \\
\Leftarrow & \qquad \text{Leibniz, } \varphi, \psi \text{ in } \mathcal{A}lg(F,E) \\
& true.
\end{array}$$

□

F. Initial algebras with laws

Let $\mathcal{C}$ be the default category. Let F be an endofunctor for which the initial algebra $\alpha = \mu F$ exists. Let $E = (T, T')$ be a law of type $(F, Id) \rightarrow (H, Id)$ for some endofunctor H. These data are fixed throughout the section.

As before, $\mathcal{A}lg(F,E)$ is the full subcategory of $\mathcal{A}lg(F)$ of algebras for which E holds. We are interested in an algebra that is initial in $\mathcal{A}lg(F,E)$; we shall denote it by $\mu(F,E)$.

36 *Example: trees continued.* Take E to be the law such that $E(e \triangledown f \triangledown \oplus)$ expresses both the neutrality of e for $\oplus$, and the associativity of $\oplus$. Then $\mu(F,E)$, if it exists, is the algebra of lists (rather than trees). Put $nil' \triangledown tip' \triangledown join' = \mu(F,E)$ and let $e \triangledown f \triangledown \oplus$ be another (F,E)-algebra. Now, by definition of initiality, the recursive equations

$$\begin{array}{lcl}
nil' \,;\, h & = & e \\
tip' \,;\, h & = & f \\
join' \,;\, h & = & I\!I h \,;\, \oplus
\end{array}$$

have precisely one solution for h, denoted $(\!|nil' \triangledown tip' \triangledown join' \rightarrow e \triangledown f \triangledown \oplus|\!)_{F,E}$. Actually, the equations imply that e is neutral for $\oplus$, and $\oplus$ is associative, at least on the range of h: see Lemma 32.

37 *Induced congruence.* We explain here informally in terms of $\mathcal{S}et$ the notion of induced congruence. Let φ be an F-algebra with carrier a, and f, g be morphisms with target a and common source; think in particular of $\varphi, (f,g) = \alpha, (T\alpha, T'\alpha)$. The pair (f,g) induces an equivalence relation p on a, namely the least equivalence relation on a that contains all (fx, gx); categorically, this is a morphism p with $f \mathbin{;} p = g \mathbin{;} p$ that has some initiality property. The target of p may be denoted $a/(f,g)$, thus $p\colon a \to a/(f,g)$.

We say that an equivalence relation p for (f,g) is a congruence for φ if p-related elements are mapped by φ to p-related results; this is almost the same as saying that p is a homomorphism from φ. Given φ and (f,g), the induced congruence is the least equivalence relation that contains all pairs (fx, gx) and is a congruence for φ; the target of p may be denoted $\varphi/(f,g)$, thus $p\colon \varphi \to_F \varphi/(f,g)$.

Thus, when α and E are given, a construction of $\mu(F,E) = \alpha/(T\alpha, T'\alpha)$ requires a construction of the congruence p for algebra α induced by $(T\alpha, T'\alpha)$. Fokkinga (1992b) gives a construction of p that simulates the well-known one for $\mathcal{S}et$, and assumes properties of the default category that are not obviously satisfied by category CPO. Since we are also interested in applications to other categories like CPO, and want to be truly general, we shall use a result from Lehmann (1978; 1980).

38 Definition. *For a pair (f,g) of morphisms with common source and with the carrier of α as their common target, the (f,g)-variety $V_F(f,g)$ is the full subcategory of $\mathcal{A}lg(F)$ of algebras φ for which $f \mathbin{;} (\!|\varphi|\!) = g \mathbin{;} (\!|\varphi|\!)$.*

Notice that $\varphi \mapsto f \mathbin{;} (\!|\varphi|\!)$ is a transformer, so that $V_F(f,g)$ equals $\mathcal{A}lg(F,E')$ where E' is the law determined by the transformers $\varphi \mapsto f \mathbin{;} (\!|\varphi|\!)$ and $\varphi \mapsto g \mathbin{;} (\!|\varphi|\!)$.

39 Theorem. (Lehmann (1980)) *For $\mathcal{C} = \mathcal{S}et$ and $\mathcal{C} = CPO$, 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 $\mathcal{C}$.*

Actually, Lehmann's theorem is more general than the above version: it involves a condition on category $\mathcal{C}$ that is satisfied by more categories than just $\mathcal{S}et$ and CPO. The following result enables us to exploit Lehmann's theorem.

40 Theorem. *Suppose that H (of the type of E) preserves epis. Then $\mathcal{A}lg(F,E)$ is a full subcategory of $V_F(E\alpha)$, and contains each φ of $V_F(E\alpha)$ for which $(\!|\varphi|\!)$ is epic in $\mathcal{C}$.*

Proof. Since both $\mathcal{A}lg(F,E)$ and $V_F(E\alpha)$ are full subcategories of $\mathcal{A}lg(F)$, we have to show, for the first claim, that each φ in $\mathcal{A}lg(F,E)$ is also in $V_F(E\alpha)$. This implication is shown as follows. Let φ be arbitrary in $\mathcal{A}lg(F)$. Then

φ is in $V_F(E\alpha)$

$\equiv$ definition 38 of $V_F(E\alpha)$, and φ is in $\mathcal{A}lg(F)$

$T\alpha \mathbin{;} (\!|\varphi|\!) = T'\alpha \mathbin{;} (\!|\varphi|\!)$

$\equiv$ TRANSFORMER at both sides (condition is satisfied: $(\!|\varphi|\!)\colon \alpha \to_F \varphi$)

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

$(*)\qquad \Leftarrow \qquad$ Leibniz

$T\varphi = T'\varphi$

$\equiv \qquad$ definition of $\mathcal{A}lg(F,E)$, and φ is in $\mathcal{A}lg(F)$

φ is in $\mathcal{A}lg(F,E)$.

For the second claim, let φ be in $V_F(E\alpha)$ such that $(\!|\varphi|\!)$ is epic in $\mathcal{C}$. Then the $\Leftarrow$ in step $(*)$ above can be strengthened to $\equiv$, since $(\!|\varphi|\!)$ is epic and H is assumed to preserve epis. Hence φ is in $\mathcal{A}lg(F,E)$ as well. □

41 Corollary. *Suppose both F and H preserve epis, and $\mathcal{C}$ satisfies the conditions of Lehmann's theorem (1980), see 39. Then $\mathcal{A}lg(F,E)$ has an initial object, denoted $\mu(F,E)$.*

Proof. By Theorem 39 $V_F(E\alpha)$ has an initial object $\alpha/E\alpha$, and $(\!|\alpha/E\alpha|\!)$ is epic in $\mathcal{C}$. (Here the conditions on F and $\mathcal{C}$ are used.) Then by Theorem 40 $\alpha/E\alpha$ is also in $\mathcal{A}lg(F,E)$, and moreover it is also initial in $\mathcal{A}lg(F,E)$. (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 $\beta = \mu(F,E)$, and write $(\!|\alpha \to \varphi|\!)_F$ as $(\!|\varphi|\!)$, and $(\!|\beta \to \varphi|\!)_{F,E}$ as $(\!|\varphi|\!)_E$.

42 Theorem. (Meertens) *Suppose (α and also) β exists, and suppose $(\!|\beta|\!)$ has a pre-inverse u . Then, for each φ in $\mathcal{A}lg(F,E)$,*

$$\begin{array}{lcl} (\!|\varphi|\!)_E &=& u\,;(\!|\varphi|\!) \\ (\!|\varphi|\!) &=& (\!|\beta|\!)\,;u\,;(\!|\varphi|\!)\,. \end{array}$$

Proof. For the first claim we argue

$(\!|\varphi|\!)_E \;=\; u\,;(\!|\varphi|\!)$

$\equiv \qquad u$ is pre-inverse of $(\!|\beta|\!)$

$u\,;(\!|\beta|\!)\,;(\!|\varphi|\!)_E \;=\; u\,;(\!|\varphi|\!)$

$\Leftarrow \qquad$ Leibniz

$(\!|\beta|\!)\,;(\!|\varphi|\!)_E \;=\; (\!|\varphi|\!)$

$\Leftarrow \qquad$ cata-Fusion

$(\!|\varphi|\!)_E\colon\ \beta \to_F \varphi$

$\equiv \qquad$ cata-Self

true.

The second claim is an immediate corollary:

$(\!|\beta|\!)\,;u\,;(\!|\varphi|\!)$

$= \qquad$ just shown

$(\!|\beta|\!) \,;\, (\!|\varphi|\!)_E$

$=$ cata-Fusion (condition is satisfied: $(\!|\varphi|\!)_E\colon\ \beta \to_F \varphi$)

$(\!|\varphi|\!)$.

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

43 *Application.* In $\mathcal{S}et$, the carrier of β consists of the $\simeq$-equivalence classes of the carrier of α , where $\simeq$ is the least equivalence that contains the pairs $((T\alpha)z, (T'\alpha)z)$ for all z . A pre-inverse u of $(\!|\alpha \to \beta|\!)$ chooses for each equivalence class a representative in the class. So the theorem says that $(\!|\beta \to \varphi|\!)_E$ at x may be computed by computing $(\!|\alpha \to \varphi|\!)$ at a representative of x instead. In this way the operational efficiency of a program may be improved.

44 *Example: trees continued.* Let $E(e \triangledown id \triangledown \oplus)$ express that $\oplus$ is an associative operation with neutral element e , and suppose that the law holds for $e \triangledown id \triangledown \oplus$. The value of $(\!|e \triangledown id \triangledown \oplus|\!)_E$ at arguments $x\,\text{join}'\,(y\,\text{join}'\,z)$ and $(x\,\text{join}'\,y)\,\text{join}'\,z$ is

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

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 $\mathit{size}\,x$, and $\mathit{size}\,(x \oplus y) = \mathit{size}\,x + \mathit{size}\,y$. (This is valid in most functional programming languages when operation $\oplus$ is the concatenation operation of lists.) Thus associativity may be exploited. More generally, let u be the function that sends $x\ \text{join}'\ y\ \text{join}'\ \dots\ \text{join}'\ z$ (with arbitrary parenthesisation) to $x\ \text{join}\ (y\ \text{join}\ (\dots\ \text{join}\ z))$ (with parenthesisation to the right). The theorem asserts that $(\!|e \triangledown id \triangledown \oplus|\!)_E = u \,;\, (\!|e \triangledown id \triangledown \oplus|\!)$, 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. In an actual program transformation u might disappear completely, namely when this transformation is but one step in a large series of steps.)

45 *Another application.* In a similar way the second claim of the theorem asserts that if φ satisfies E , then "within the argument" of $(\!|\varphi|\!)$ operation α may be manipulated as if it satisfies E , that is, $T\alpha \,;\, (\!|\varphi|\!) = T'\alpha \,;\, (\!|\varphi|\!)$. This is shown as follows.

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

$\equiv$ second claim of the theorem: $(\!|\varphi|\!) = (\!|\beta|\!) \,;\, u \,;\, (\!|\varphi|\!)$

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

$\Leftarrow$ Leibniz

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

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

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

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

true.

G. Each colimit is an initial lawful algebra

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

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

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

Now take $F = \underline{a}$, the constant functor mapping any morphism onto id_a. Take

$$E \;=\; (T,T') \qquad \text{with} \qquad Tq \;=\; f\,;q \qquad \text{and} \qquad T'q \;=\; g\,;q$$

for each $q\colon\ Fb \to b \quad = \quad a \to b$. Then, in the notation of Theorem 19, $T = \underline{f};I$ and similarly $T' = \underline{g};I$, and so E is a law (by that same theorem). Further, Ep holds, and $p\colon\ F(\text{tgt}\,p) \to \text{tgt}\,p$. So p is an (F,E)-algebra. To show the initiality of p we shall prove cata-CHARN, deriving along the way a definition for the required $(\!|p \to q|\!)$.

$$\begin{array}{ll}
 & x\colon\ p \to_F q \\
\equiv & \qquad \text{definition } \to_F \text{ and } F = \underline{a} \\
 & p\,;x = q \\
\equiv & \qquad \text{coequaliser-CHARN, noting that } f\,;q = g\,;q \\
 & x = p\backslash q \\
\equiv & \qquad \textbf{defining } (\!|p \to q|\!) = p\backslash q \\
 & x = (\!|p \to q|\!).
\end{array}$$

47 *Colimits.* 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". First we give a formalisation of colimit that suits the present purpose well.

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

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

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

48 *The construction.* Take $F = \underline{\Sigma D}$, where $\Sigma D =$ (the carrier of) the sum of all objects in D. Similarly to the Trees example each F-algebra $\delta'\colon\ \Sigma D \to d$ can be written as $\delta' = \nabla(a \text{ in } D :: \mathit{in}_a \,;\, \delta')$. We design E such that $E\delta$ equivales (a) above:

$$E \quad = \quad \text{the conjunction of the laws } (T_a, T'_{f,b}) \text{ for all } f\colon\ a \to b \text{ in } D$$

where

$$\begin{aligned} T_a\delta' &= \mathit{in}_a \,;\, \delta' \\ T'_{f,b}\delta' &= f \,;\, \mathit{in}_b \,;\, \delta' \,. \end{aligned}$$

Indeed, $\delta' = \nabla(a :: \delta_a)$ is an (F,E)-algebra iff $\delta = (a :: \delta_a)$ is a cocone for D. Moreover, by Theorem 19 T_a and $T'_{f,b}$ are transformers, so that $(T_a, T'_{f,b})$ is a law, and by Theorem 26 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 $\mathcal{A}lg(F,E)$. To show this let $\delta' = \nabla(a :: \delta_a)$ be an arbitrary (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 $(\!|\gamma' \to \delta'|\!)$ meets the requirement of cata-Charn:

$$\begin{aligned} & x\colon\ \gamma' \to_F \delta' \\ \equiv \quad & \qquad \text{definition } \to_F \\ & \gamma' \,;\, x = Fx \,;\, \delta' \\ \equiv \quad & \qquad \text{definition } F \text{ as a constant functor} \\ & \gamma' \,;\, x = \delta' \\ \equiv \quad & \qquad \text{definition } \gamma' \text{ and } \delta' \text{ and sum} \\ & \forall a \text{ in } D ::\quad \gamma_a \,;\, x = \delta_a \\ \equiv \quad & \qquad \text{colimit-Charn: (b) above} \\ & x = \gamma\backslash\delta \\ \equiv \quad & \qquad \text{definition `} (\!|\gamma' \to \delta'|\!) \text{'} \\ & x = (\!|\gamma' \to \delta'|\!). \end{aligned}$$

So γ' is initial indeed.

H. Equational specification of datatypes

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

$$\begin{aligned} \varphi &= \mathit{empty} \triangledown \mathit{push} &&: \quad 1 + \mathit{nat} \times \mathit{stk} \to \mathit{stk} &&= F\mathit{stk} \to \mathit{stk} \\ \psi &= \mathit{isempty} \vartriangle \mathit{top} \vartriangle \mathit{pop} &&: \quad \mathit{stk} \to \mathit{bool} \times \mathit{nat} \times \mathit{stk} &&= \mathit{stk} \to G\mathit{stk} \end{aligned}$$

where stk is the set of the stack values, and apparently $F = \underline{1} + \underline{\mathit{nat}} \times \mathit{Id}$ and $G = \underline{\mathit{bool}} \times \underline{\mathit{nat}} \times \mathit{Id}$. (We could consider stacks over a for arbitrary a, but we wish to avoid the complications involved.) Often for such "datatypes" some law E is imposed that defines the ψ-part in terms of the φ-part. For stacks the laws are

$$\begin{aligned} \mathit{empty} \,;\, \mathit{isempty} &= \mathit{true} \\ \mathit{push} \,;\, \mathit{isempty} &= \mathit{false} \end{aligned}$$

$$\begin{array}{lcl} push \mathbin{;} top & = & exl \\ push \mathbin{;} pop & = & exr\ . \end{array}$$

Written as two equations:

$$\begin{array}{lcl} empty \mathbin{;} \psi & = & id_1 \ \mathbin{;}\ true \vartriangle \ldots \vartriangle \ldots \\ push \mathbin{;} \psi & = & id_{nat} \times \psi \ \mathbin{;}\ (! \mathbin{;} false) \vartriangle exl \vartriangle (exr \mathbin{;} ex_{3,1} \vartriangle ex_{3,2} \mathbin{;} push) \end{array}$$

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

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

for some transformer T of type $(F, Id) \to (FG, G)$. Theorem 51 below asserts that for a law of this form, with arbitrary transformer T, the ‘datatype’ (φ, ψ) is isomorphic to the initial F-algebra (the φ-part) to which $(\!|T\varphi|\!)$ (the ψ-part) is added as a derived operation. Since for the F above the initial F-algebra is known as the cons-lists over nat, we find that the datatype of stacks over nat is semantically just the algebra of cons-lists over nat with some additional derived operations, “destructors” in this case.

50 *Bialgebras.* Fix a default category, and let F, G be endofunctors. An F, G-bialgebra is: a pair (φ, ψ) with $\varphi\colon Fa \to a$ and $\psi\colon a \to Ga$, for some a called the carrier of the bialgebra. So, the pair consists of an algebra and a co-algebra with the same carrier. Given F, G-bialgebras (φ, ψ) and (χ, ω), we say f is a homomorphism from (φ, ψ) to (χ, ω) if: $f\colon \varphi \to_{F,Id} \chi$ and $f\colon \psi \to_{Id,G} \omega$, using the notation of dialgebras. This determines a category $\mathcal{B}iAlg(F, G)$: the objects are F, G-bialgebras, the morphisms are F, G-bialgebra homomorphisms, and the composition and identities are inherited from the default category.

Actually, a bialgebra is merely a particular dialgebra: defining $(F \vartriangle G)x = (Fx, Gx)$ we have $\mathcal{B}iAlg(F, G) = \mathcal{D}iAlg(F \vartriangle Id, Id \vartriangle G)$. Hence there are no new concepts involved, but only specialisations of known ones.

The notion of transformer and law makes sense for bialgebras, thus. If E is such a law we let $\mathcal{B}iAlg(F, G; E)$ denote the full subcategory of $\mathcal{B}iAlg(F, G)$ of those bialgebras that satisfy E.

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

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

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

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 a F, G-bialgebra for which E holds. We shall show that

$$x\colon\ (\alpha, (\!|T\alpha|\!)) \to_{\mathcal{B}iAlg(F,G)} (\varphi, \psi) \qquad \equiv \qquad x = (\!|\varphi|\!)$$

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

$$\begin{array}{lll}
 & & x\colon\ (\alpha, (\!|T\alpha|\!)) \to_{\mathcal{B}iAlg(F,G)} (\varphi, \psi) \\
 & \equiv & \quad \text{definition } \mathcal{B}iAlg(F,G) \\
 & & x\colon\ \alpha \to_F \varphi \quad \wedge \quad x\colon\ (\!|T\alpha|\!) \to_{G,Id} \psi \\
 & \equiv & \quad \text{cata-}\textsc{Charn} \\
 & & x = (\!|\varphi|\!) \quad \wedge \quad (\!|\varphi|\!)\colon\ (\!|T\alpha|\!) \to_{G,Id} \psi \\
(*) & \equiv & \quad \text{below} \\
 & & x = (\!|\varphi|\!).
\end{array}$$

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

$$\begin{array}{ll}
 & (\!|\varphi|\!)\colon\ (\!|T\alpha|\!) \to_{G,Id} \psi \\
\equiv & \quad \text{definition } \to_{G,Id} \\
 & (\!|T\alpha|\!) \,;\, G(\!|\varphi|\!) = (\!|\varphi|\!) \,;\, \psi \\
\equiv & \quad \text{rhs: cata-}\textsc{Fusion} \text{ (condition `} \psi\colon\ \varphi \to_F T\varphi \text{' equivales } E(\varphi, \psi) \text{)} \\
 & (\!|T\alpha|\!) \,;\, G(\!|\varphi|\!) = (\!|T\varphi|\!) \\
\Leftarrow & \quad \text{cata-}\textsc{Fusion} \\
 & T\alpha \,;\, G(\!|\varphi|\!) = FG(\!|\varphi|\!) \,;\, T\varphi \\
\Leftarrow & \quad \textsc{Transformer} \\
 & \alpha \,;\, (\!|\varphi|\!) = F(\!|\varphi|\!) \,;\, \varphi \\
\equiv & \quad \text{cata-}\textsc{Self} \\
 & \mathit{true}.
\end{array}$$

□

Similarly one may specify an $F{+}G$-algebra $\varphi \triangledown \psi$ 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.

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

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

Then $\alpha \triangledown T\alpha$ is initial in $\mathcal{A}lg(F + G; E)$.

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

$$\begin{array}{ll}
 & x\colon\ \alpha \triangledown T\alpha \to_{F+G} \varphi \triangledown \psi \\
\equiv & \quad \text{definition } \to_{F+G} \\
 & x\colon\ \alpha \to_F \varphi \quad \wedge \quad x\colon\ T\alpha \to_G \psi \\
\equiv & \quad \text{cata-}\textsc{Charn}
\end{array}$$

$$\begin{array}{lll}
 & & x = (\!|\varphi|\!) \quad \wedge \quad (\!|\varphi|\!) \colon T\alpha \to_G \psi \\
(*) & \equiv & \qquad \text{below} \\
 & & x = (\!|\varphi|\!) .
\end{array}$$

Step $(*)$ is verified as follows.

$$\begin{array}{ll}
 & (\!|\varphi|\!) \colon T\alpha \to_G \psi \\
\equiv & \qquad \text{law } E \text{ holds for } \varphi \triangledown \psi \\
 & (\!|\varphi|\!) \colon T\alpha \to_G T\varphi \\
\Leftarrow & \qquad \textsc{Transformer} \\
 & (\!|\varphi|\!) \colon \alpha \to_F \varphi \\
\equiv & \qquad \text{cata-}\textsc{Self} \\
 & \mathit{true} .
\end{array}$$

□

It is straightforward to combine both theorems, and generalise 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.

I. Conclusion

We have proposed a semantical, categorical, characterisation of what a term (as used in conditional equations) is: the TRANSFORMER property. The property is almost as simple as the defining property of functor, and a mapping that satisfies the TRANSFORMER 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.

The notion of transformer seems to allow for a great simplification of the theory of equational specification of datatypes as far as only semantic aspects are concerned. Compare for example the exposition in Section H with current literature on 'equational specification of datatypes' such as Ehrig and Mahr's book (1985). Our formalism 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. The use of transformer avoids the introduction of non-semantic aspects. Much more in this area can be done.

In this paper, the development and application of the notion of transformer has only begun, and much remains to be investigated. At the category theory side the relation between monad theory and our method of dealing with lawfull algebras calls for clarification, since monad theory is another way of dealing with (single sorted) algebras with equations. At the computing science side more applications are waiting to be discovered. We mention two that have already been discovered.

First, Erik Meijer (1994) describes the derivation of compilers from a denotational semantics. The notion of transformer turns up, here, to structure the relationship of various mappings like compiler, interpreter, and proper semantics. He uses the TRANSFORMER property to calculate one from the other.

Second, thanks to the formalisation of the notion of law, one can now formulate conjectures, statements and proofs about 'laws in general'. For example, consider the operation of *lifting*: the binary addition $+\colon I\!I\, nat \to nat$ is lifted to the binary $+'\colon I\!I(a \to nat) \to (a \to nat)$ defined by: $(f +' g)x = fx + gx$. It is easy to see that $+'$ is associative because $+$ is, and $+'$ is commutative since $+$ is. In general, any operation $\varphi\colon Fb \to c$ can be lifted to an operation $\varphi'\colon F(a \to b) \to (a \to c)$. One may now conjecture that lifting preserves the validity of all algebraic laws. In order to formally state this conjecture, one needs a formalisation of the notion of law. Meertens and Van der Woude (in unpublished work) have been able to formally prove this conjecture — using the notion of transformer.

Acknowledgement. The Constructive Algorithmics Club 1989-1991 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 G and part of Theorems 23 and 42. Erik Meijer brought Lehmann (1978) to my attention. Ross Paterson suggested several improvements in the presentation, and also some further generalisations.

I wish to thank one referee for suggesting numerous minor improvements in the formulations, and another one for suggesting a few major lines of further research.

References

M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall, 1990.

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

M.M. Fokkinga. Calculate categorically! *Formal Aspects of Computing*, 4(4):673–692, 1992.

M.M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, dept Comp Sc, Enschede, The Netherlands, 1992.

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

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.

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

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

G. Malcolm. *Algebraic Data Types and Program Transformation*. PhD thesis, University of Groningen, The Netherlands, 1990.

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

E.G. Manes. *Algebraic Theories*, volume 26 of *Graduate Text in Mathematics*. Springer Verlag, 1987.

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

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.

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.

E. Meijer, M.M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *FPCA91: Functional Programming Languages and Computer Architecture*, volume 523 of *Lect. Notes in Comp. Sc.*, pages 124–144. Springer Verlag, 1991.

E. Meijer. More Advice on Proving Compilers Correct: Improving Correct Compilers. Submitted for publication, 1994.

B.C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, Cambridge, Ma, 1991.

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