1991

M.M. Fokkinga

Calculate categorically!

Computer Science/Department of Algorithmics and Architecture Report CS-R9132 June

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

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

Calculate Categorically!

Maarten M. Fokkinga*

'Diagram chasing' is an established proof technique in Category Theory. Algebraic *calculation* is a good alternative; made possible thanks to a *notation* for various 'unique arrows' and a suitable *formulation* of 'initiality', and the properties brought forward by initiality.

1991 CR Classification System: D.1.1, F.3.2-3.
1980 Mathematics Subject Classification: 18A05, 69D11, 69F32, 69F33.
Key Words and Phrases: category theory, diagram chasing, initiality, algebraic calculation, functional programming.

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

Report CS-R9132
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1 Introduction

Category Theory [20] is a field of mathematics that seeks to discuss and unify many concepts occurring in mathematics. In the last decade it has proved useful for computing science as well; this may be evident from the rapidly growing number of conferences and publications with 'Category Theory' and 'Computer Science' in their title [16, 3, 26, 27, 13, 29, and so on]. Not only is category theory helpful to formalise and prove results for theoretical aspects of computing science, like lambda calculus theory, denotational semantics, and fundamentals of algebraic specification, but also to formalise and prove results for practical aspects like language design and implementation (e.g., Hagino [15, 14], Reynolds [28] and Cousineau et al [6]) and program derivation (e.g., Malcolm [21, 22] and Fokkinga and Meijer [10]).

In this paper we pave the way for a style of proof that is an alternative to the conventional one in Category Theory: *calculation* instead of *diagram chasing*. In effect, it is a form of Functional Programming. Let us explain the key-words.

Category Roughly said, a category is just a collection of arrows with the closure property that "composition of two arrows f and g with $\text{target}(f) = \text{source}(g)$, is an arrow again." Thus, a mathematical structure, when studied categorically, must be modeled as a system of arrows. This may pose serious problems to the newcomer; Arbib and Manes [1] teach how to think in terms of arrows. The prominent rôle of arrows invites to use pictures containing (a lot of) arrows, so-called diagrams, as a tool in categorical proofs. Take any book on category theory and you'll see what I mean! The conventional style of proof is diagram chasing (explained below); we offer an alternative: algebraic calculation. To do so, we give a systematic treatment of the calculation properties brought forward by *initiality*, and show them in action on a variety of examples. Initiality is a categorical concept by which many, very many mathematical constructions can be characterised.

Diagram chasing There are several reasons why diagrams are so popular among categoricians, and one has to face all of them when judging the relative merits of an alternative style of proof. Let us consider (all?) four of these reasons.

- *Typing.* A picture may clearly indicate which arrows have a common source or target, much more so than a linear listing of the arrows with the source and target given for each of them.

  Remark. The need for an overview of the sources and targets of the arrows is partly caused by the notation $f : a \to b$ and $g : b \to c$ to indicate the source and target (called *typing*), and the notation $g \circ f$ for their composition. We choose the notation $f;g$ for composition, so that $f;g : a \to c$ falls out naturally. (An alternative would be to use the notation $f : a \leftarrow b$ and $g : b \leftarrow c$ so that $f \circ g : a \leftarrow c$. However, this direction of the arrows conflicts with the conventional left-to-right direction of arrows and input-to-output in our Western Culture.)

- *Naming.* Initiality means that for certain pairs of source and target there is precisely one arrow in between. A picture is a suitable tool to indicate such an arrow, typically by a dashed line, and to attach a name to it so that you can refer to it in the text. Without pictures one usually introduces such an arrow by a phrase like "Let f be the unique arrow from *this* to *that* that exists on account of initiality of *such-and-so*."

  Remark. We shall use a standard notation for various 'unique' arrows; the notation will clearly suggest the source and target, as well as some other properties. Thanks to the availability of a notation there is no need to interrupt an argument or calculation for a verbose introduction of such an arrow: you can just denote it.

- *Commuting diagrams.* Equality of arrows can be indicated pictorially if, by convention, in the picture any two (composite) arrows with the same source and the same target are equal. Thus $f\,;g = h$ appears as a triangle, and $f\,;g = h\,;j$ as a quadrangle. This convention is called *commutativity of the diagram.* A commuting composite diagram is a very economical way of showing several equalities simultaneously without duplication of subterms that denote arrows. Moreover, it is easy to indicate all sources and targets explicitly in the diagram.

  Remark. It would be absurd if triangles and quadrangles are clearer than formulas of the shape $f\,;g = h$ and $f\,;g = h\,;j$. In fact, even complicated formulas like $f_0\,;f_1\,;\ldots;f_{m-1} = g_0\,;g_1\,;\ldots;g_{n-1}$ are not more (and no more) understandable by drawing the left-hand side and right-hand side as a stretched, possibly wriggled quadrangle. In almost any diagram there is one equation of interest (the theorem) and the other equations in the diagram are just auxiliary, for use in the proof only; in that case there is no need to display them all at once (if an alternative proof does the job). Similarly, the information about source and target of each term occurring in a proof is often not helpful for understanding the main equation or verifying the proof steps, as we will see.

- *Diagram chasing.* Pasting several commuting diagrams together along common arrows gives a commuting diagram as result. It is an easy, visual, reliable style of proving equality of arrows. This is the real reason why there appear composite diagrams that are more complicated than the simple polygons.
  It is particularly easy to extend a diagram with an arrow; in a calculation one would have to *copy* the equation obtained thus far, and transform that a little.

  Remark. This use of diagrams may be quite helpful when conducting a proof on a blackboard, with an eraser at hand. (Also, it lends itself well for presentations with an overhead projector, using overlays.) However, in the final picture the history is completely lost. It is then just a puzzle, called *diagram chasing*, to find out what arrows exist for what reason, and what subdiagrams commute on what grounds. Moreover, it is even much harder to read off from the final diagram for what reason a certain arrow is the only one possible that makes a certain subdiagram commute. It is all this implicit information that is so clearly present in the calculations below.

So far for the reasons to use pictures and diagram chasing, and some objections.

Calculation The pasting together of two diagrams is calculationally rendered by transitivity of equality: if the two diagrams "prove" $f = g$ and $g = h$ respectively, and they are pasted together along arrow g, then the resulting diagram proves the new equality $f = h$. There is no problem in presenting this step in a calculation; for example:

$$\begin{array}{ll} & f \\ = & \quad \text{reasoning of one diagram} \\ & g \\ = & \quad \text{reasoning of the other diagram} \\ & h \end{array}$$

or, alternatively,

$$\begin{array}{ll} & f = g \ \wedge \ g = h \\ \Rightarrow & \quad \text{transitivity} \\ & f = h. \end{array}$$

More important is a formalisation of initiality that lends itself to such a calculational, equational reasoning. By definition, a is initial if for each target b there is precisely one arrow from a to b. Formally, a is initial if

$$\exists(x :: \ x : a \to b \ \wedge \ \forall(y :: \ y : a \to b \ \Rightarrow \ y = x)),$$

for all b. It is the presence of the existential quantifier (and the universal one in its scope) that hinders equational reasoning. An equivalent formalisation of initiality of a reads: there exists a function $\mathcal{F}$ such that

$$x : a \to b \ \equiv \ x = \mathcal{F}b,$$

for all x and b. Indeed, substituting $x = \mathcal{F}b$ gives $\mathcal{F}b : a \to b$ (there is at least one arrow), and the implies part of the equivalence gives that there is at most one arrow. We shall see that this formalisation is the key to calculational reasoning (of the second kind illustrated above). The use of equivalences to characterise initiality (and more generally, universality) has been thoroughly advocated by Hoare [16]. As far as we know, Malcolm [21] was the first to use this style of reasoning for the derivation of functional programs over initial algebras.

The format of a calculation We present a calculation in the way we have actually derived it (or would like to have derived it). In general, we start with the main proof obligation (an equation), and reduce it step by step to simpler proof obligations (equations), until we finally arrive at **true**. (When calculating with terms denoting arrows, we usually start with the more complicated form, and transform it step by step to the simpler one.) This style of conducting and reading proofs requires some exercising to get used to; once mastered it turns out to be an effective way of working. Dijkstra and Scholten [7] discuss this in detail, and attribute the calculational format to van Gasteren [11] and Feijen.

Functional programming In this paper all arrows (in the sequel called morphisms) may be interpreted as typed total functions; there is simply no axiom for the category under consideration, that prohibits this interpretation. Therefore one may interpret our activity as functional programming, though for a bit unusual specifications. The combinations and transformations of morphisms (functions) are fully in the spirit of Backus [2] and Meertens [24]. One should note that nowhere in this paper a morphism (function) is applied to an argument; it is just by composing functions in various ways that new functions are formed and equalities are proved. The absence of restrictions on combining functions (except for typing constraints) has often been claimed to be a major benefit of functional programming, e.g., by Hughes [17].

Historical remark Our interest was in the development of a calculus for the derivation of algorithms from a specification, as proposed by Meertens [24] and Bird [4, 5]. Category theory provides a suitable medium to formalise the notion of datatype, as shown by Lehmann and Smith [19], Manes and Arbib [23], and many others. Then Malcolm [21], Spivey [30], and others showed that many of the properties that are *actually* used in program derivations can be inferred from the categorical description of a datatype. It is from here a small step to apply the calculational style of program derivation more generally to category theory itself.

The overall acceptance of diagram chasing is presumably the cause that this style of deriving categorical properties is relatively unknown. Indeed, only recently books and papers have appeared on category theory in which equational reasoning is explicitly strived for; for example by Lambek and Scott [18], Hoare [16].

* * *

The remainder of the paper is organised as follows. In the next section we explain our notation and terminology, define some lesser known categorical concepts, and discuss initiality. We assume that the reader is familiar with the basics of category theory; Goldblatt [12, Chapters 2,3,9] and Pierce [25] give a good introduction. Then we specialise the laws for initiality to products and coproducts in Section 3, to initial algebras in Section 4, to coequalisers and kernel pairs in Section 5, and finally to colimits in general in Section 6. Each section contains one or more examples of a calculation for the derivation of a well-known result.

The proof of the pudding is in the eating: the categorician should compare our calculations with the usual pictorial proofs, and pay attention to the precision, conciseness, and clarity with which various steps in the proofs are stated, and to the absence of verbose or pictorial introductions of various unique arrows.

2 Preliminaries and Initiality

We use the word *morphism* instead of *arrow*.

Nomenclature Variables $\mathbf{A}, \mathbf{B}, \mathbf{C}$ denote categories, and capital letters vary over functors. Formula $x : a \to_{\mathbf{A}} b$ means that x is a morphism in $\mathbf{A}$ with source a and target b ($\mathrm{src}_{\mathbf{A}}\ x = a$ and $\mathrm{tgt}_{\mathbf{A}}\ x = b$). In each section, the category that underlies the construction of another, or several others, is denoted $\mathbf{C}$, and is called the base category. Unless stated otherwise, a morphism is a morphism in $\mathbf{C}$, and similar for objects. We shall mostly suppress mentioning of the base category $\mathbf{C}$, certainly where it would occur as a subscript. In particular, we almost always write $\to$ for $\to_{\mathbf{C}}$. By default, $a, b, c, \ldots$ vary over objects in $\mathbf{C}$, and $f, g, h, j, \ldots, p, q, \ldots, x, y, \ldots$ over morphisms in $\mathbf{C}$. We denote composition of the base category (and all the categories that inherit the composition of the base category) in diagramatic order: if $x : a \to b$ and $y : b \to c$ then $x ; y : a \to c$. Composition of functors and other mappings is denoted by juxtaposition: $(FG)f = F(Gf)$.

Category **Set** has as objects sets, and as morphisms typed total functions. We shall nowhere impose an axiom on base category $\mathbf{C}$ that prohibits to interpret it as **Set**.

Remark It is the morphisms in a category that are important; the objects only serve an auxiliary role, mainly for the well-formedness of morphism composition. In each category $\mathbf{A}$, including those of functors and so on, each morphism has a unique source and target, so that you need not mention them explicitly; they can be retrieved from the morphism itself by the functions $\mathrm{src}_{\mathbf{A}}(\_)$ and $\mathrm{tgt}_{\mathbf{A}}(\_)$. Thus we view a functor mainly as a mapping of morphisms; its action on objects can then be derived. In the same vein, we shall define concepts in terms of morphisms as much as possible, and suppress the auxiliary role of objects when that can be derived or is clear from the context. □

Categories built upon C Often an interesting construction in $\mathbf{C}$ can be characterised by initiality in a category $\mathcal{C}$ built upon $\mathbf{C}$. We say $\mathcal{C}$ is *built upon* $\mathbf{C}$ if each morphism of $\mathcal{C}$ is a -special- morphism in $\mathbf{C}$ and $\mathcal{C}$'s composition is that of $\mathbf{C}$. So, $\mathcal{C}$ is fully determined by defining its objects and morphisms. Here are some examples that we'll meet in the sequel; skip the description upon first reading. (The specialist may recognise $\bigvee(D)$ as the category of *cocones* for the *diagram* D. Dually, the category of *cones* for D is denoted $\bigwedge(D)$.)

Category $\bigvee(\vec{a})$, where $\vec{a}$ is an n-tuple of objects in $\mathbf{C}$. An object in $\bigvee(\vec{a})$ is an n-tuple of confluent morphisms in $\mathbf{C}$ with the objects $\vec{a}$ as sources, as suggested by the symbol $\bigvee$ in case $n = 2$. Let $\vec{f}$ and $\vec{g}$ be such objects; then a morphism from $\vec{f}$ to $\vec{g}$ in $\bigvee(\vec{a})$ is a morphism x in $\mathbf{C}$ satisfying $f_i ; x = g_i$ for each index i of the n-tuple. It follows that $x : \mathrm{tgt}\, \vec{f} \to \mathrm{tgt}\, \vec{g}$. (For $n = 1$, category $\bigvee(a)$ is known as the slice category 'Under a', usually denoted $a{\downarrow}\mathbf{C}$ or $a/\mathbf{C}$.)

Category $\bigvee(f \| g)$, where f and g are morphisms in $\mathbf{C}$ with a common source and a common target. An object in $\bigvee(f \| g)$ is a morphism p for which $f ; p = g ; p$. Let p and q be such objects; then a morphism from p to q in $\bigvee(f \| g)$ is a morphism x in $\mathbf{C}$ satisfying $p ; x = q$. (So, $\bigvee(f \| g)$ is a full subcategory of $\bigvee(a)$ where $a = \mathrm{tgt}\, f = \mathrm{tgt}\, g$.)

Category $\mathrm{Alg}(F)$, where F is an endofunctor on $\mathbf{C}$. An object ϕ of $\mathrm{Alg}(F)$ is a morphism $\phi : Fa \to a$ in $\mathbf{C}$, for some a called the *carrier* of ϕ. Let ϕ and ψ be such objects; then a morphism from ϕ to ψ in $\mathrm{Alg}(F)$ is a morphism x in C satisfying

$\phi ; x = Fx ; \psi$. It follows that $x : \text{carrier}\,\phi \to \text{carrier}\,\psi$, and $\text{carrier}\,\phi = \text{tgt}\,\phi$. An object ϕ is called an *F-algebra*, and a morphism x is called an *F-homomorphism*. We shall later explain this in more detail.

Initiality

Let $\mathbf{A}$ be a category, and a an object in $\mathbf{A}$. Then a is *initial in* $\mathbf{A}$ if:

$$x : a \to_{\mathbf{A}} b \quad \equiv \quad x = (\!|a \multimap b|\!)_{\mathbf{A}} \qquad \text{CHARN}$$

Here $(\!|a \multimap b|\!)_{\mathbf{A}}$ is just a notation, a name, for a morphism (depending on $\mathbf{A}, a$ and b), and all free variables in the line are understood to be universally quantified, except those that have been introduced in the immediate context ($\mathbf{A}$ and a in this case). 'CHARN' is mnemonic for Characterisation. The $\Rightarrow$ part of CHARN says that each morphism x with a as its source, is uniquely determined by its target b (if it exists at all). From the $\Leftarrow$ part, taking $x := (\!|a \multimap b|\!)_{\mathbf{A}}$, it follows that for each b there is a morphism from a to b. Thus $(\!|\ |\!)$ is a standard name for the unique morphisms from a. Often there is a more specific notation that better suggests the resulting properties (see the following sections).

Of course, when $\mathbf{A}$ is clear from the context we omit the indication of $\mathbf{A}$. It often happens that one initial object in $\mathbf{A}$ is fixed, and in that case $(\!|b|\!)$ abbreviates $(\!|a \multimap b|\!)$. (The usual notation for $(\!|b|\!)_{\mathbf{A}}$ is $!_b$. The $!$-notation doesn't work well for categories built upon $\mathbf{A}$.) Finality is dual to initiality: an object a is *final* if for any object b there exists precisely one morphism from b to a. The default notation for this unique morphism is $[\!|b \multimap a|\!]_{\mathbf{A}}$.

Let us now give some consequences of CHARN. By a substitution for x such that the right-hand side becomes true we find SELF, and by a substitution for b, x such that the left-hand side becomes true we find ID:

$$(\!|a \multimap b|\!)_{\mathbf{A}} : a \to_{\mathbf{A}} b \qquad \text{SELF}$$
$$id_a = (\!|a \multimap a|\!)_{\mathbf{A}} \qquad \text{ID}$$

Next we have the Uniqueness and Fusion property (still assuming a initial in $\mathbf{A}$):

$$x, y : a \to_{\mathbf{A}} b \quad \Rightarrow \quad x = y \qquad \text{UNIQ}$$
$$x : b \to_{\mathbf{A}} c \quad \Rightarrow \quad (\!|a \multimap b|\!)_{\mathbf{A}} ; x = (\!|a \multimap c|\!)_{\mathbf{A}} \qquad \text{FUSION}$$

The proof of UNIQ is left to the reader. For FUSION we argue (suppressing $\mathbf{A}$ and a):

$$\begin{array}{ll} & (\!|b|\!) ; x = (\!|c|\!) \\ \equiv & \quad \text{CHARN}[b, x := c, (\!|b|\!) ; x] \\ & (\!|b|\!) ; x : a \to c \\ \Leftarrow & \quad \text{composition} \end{array}$$

$$
\begin{array}{ll}
& (\!|b|\!) : a \to b \quad \wedge \quad x : b \to c \\
\equiv & \quad \text{SELF, and premise} \\
& \mathsf{true}.
\end{array}
$$

These five laws become much more interesting in case category $\mathbf{A}$ is built upon another one, and $\to_{\mathbf{A}}$ is expressed as one or more equations in the underlying category. In particular the importance of law FUSION cannot be over-emphasised; we shall use it quite often. If the statement $x : b \to_{\mathbf{A}} c$ boils down to the equation $c = b\,;x$ (which is the case when $\mathbf{A} = \bigvee(a)$), law FUSION can be formulated as an unconditional equation (by substituting $c := b\,;x$ in the consequent, giving $(\!|b|\!)\,;x = (\!|b\,;x|\!)$). In the case of initial algebras UNIQ captures the pattern of proofs by induction that two functions x and y are equal; in several other cases UNIQ asserts that a collection of morphisms is jointly epic.

Application Here is a first example of the use of these laws: proving that an initial object is unique up to a unique isomorphism. Suppose that both a and b are initial. We claim that $(\!|a \multimap b|\!)$ and $(\!|b \multimap a|\!)$ establish the isomorphism and are unique in doing so. By SELF they have the correct typing. We shall show

$$x = (\!|a \multimap b|\!) \;\wedge\; y = (\!|b \multimap a|\!) \quad \equiv \quad x\,;y = id_a \;\wedge\; y\,;x = id_b$$

that is, their composition is the identity, and conversely the identities can be factored only in this way. We prove both implications of the equivalence at once.

$$
\begin{array}{ll}
& x = (\!|a \multimap b|\!) \quad \wedge \quad y = (\!|b \multimap a|\!) \\
\equiv & \quad \text{CHARN} \\
& x : a \to b \quad \wedge \quad y : b \to a \\
\equiv & \quad \text{composition} \\
& x\,;y : a \to a \quad \wedge \quad y\,;x : b \to b \\
\equiv & \quad \text{CHARN} \\
& x\,;y = (\!|a \multimap a|\!) \quad \wedge \quad y\,;x = (\!|b \multimap b|\!) \\
\equiv & \quad \text{ID} \\
& x\,;y = id_a \quad \wedge \quad y\,;x = id_b.
\end{array}
$$

The reader may show $(\!|a \multimap b|\!)\,;(\!|b \multimap a|\!) = id_a$ alternatively using ID, FUSION, and SELF in that order. (This gives a nice proof of the weaker claim that initial objects are isomorphic.)

3 Products and Coproducts

Products and coproducts are the categorical formalisation of what is known in **Set** as cartesian product and disjoint union. (In other categories products and coproducts may get a different interpretation.)

Let a, b be objects of $\mathbf{C}$. By definition, a *coproduct* of a and b is an initial object in $\bigvee(a,b)$; it may or may not exist. Let inl, inr be a coproduct of a and b; their common target is often denoted $a+b$. We write $f \triangledown g$ instead of $(\!|inl, inr \rightarrow f, g|\!)_{\bigvee(a,b)}$, thus suppressing the dependency on a, b and inl, inr. (The usual categorical notation is $[f,g]$, and its type is $a+b \rightarrow c$ provided that $f : a \rightarrow c$ and $g : b \rightarrow c$.) Working out $\rightarrow_{\bigvee(a,b)}$ in terms of equations in $\mathbf{C}$, morphisms inl, inr and operation $\triangledown$ are determined ("up to isomorphism") by law CHARN, and consequently also satisfy the other laws.

$$inl\,;x = f \;\wedge\; inr\,;x = g \quad\equiv\quad x = f \triangledown g \qquad \triangledown\text{-CHARN}$$
$$inl\,;f \triangledown g = f \;\wedge\; inr\,;f \triangledown g = g \qquad \triangledown\text{-SELF}$$
$$inl \triangledown inr = id \qquad \triangledown\text{-ID}$$
$$inl\,;x = inl\,;y \;\wedge\; inr\,;x = inr\,;y \quad\Rightarrow\quad x = y \quad (inl, inr \text{ jointly epic}) \qquad \triangledown\text{-UNIQ}$$
$$f\,;x = h \;\wedge\; g\,;x = j \quad\Rightarrow\quad f \triangledown g = h \triangledown j \qquad \triangledown\text{-FUSION}$$

Law FUSION may be simplified to an unconditional law by substituting $h, j := f\,;x,\ g\,;x$:

$$f \triangledown g\,;x = (f\,;x) \triangledown (g\,;x) \qquad \triangledown\text{-FUSION}$$

Similar simplifications will occur often in the sequel.
The reader may check that her favourite 'implementation' of disjoint union in **Set**, namely 'inleft' inl, 'inright' inr, and 'case' $\triangledown$, satisfies these laws.

Products are, by definition, dual to coproducts. Let $outl, outr$ be a product of a and b; its common source is often denoted $a \times b$. We write $f \vartriangle g$ for $[\![f, g \rightarrow outl, outr]\!]_{\bigwedge(a,b)}$. (The usual categorical notation is $\langle f, g \rangle$, and its type is $c \rightarrow a \times b$ provided that $f : c \rightarrow a$ and $g : c \rightarrow b$.) The laws for $outl$, $outr$ and $\vartriangle$ read now:

$$x\,;outl = f \;\wedge\; x\,;outr = g \quad\equiv\quad x = f \vartriangle g \qquad \vartriangle\text{-CHARN}$$
$$f \vartriangle g\,;outl = f \;\wedge\; f \vartriangle g\,;outr = g \qquad \vartriangle\text{-SELF}$$
$$outl \vartriangle outr = id \qquad \vartriangle\text{-ID}$$
$$x\,;outl = y\,;outl \;\wedge\; x\,;outr = y\,;outr \quad\Rightarrow\quad x = y \qquad \vartriangle\text{-UNIQ}$$
$$x\,;f \vartriangle g = (x\,;f) \vartriangle (x\,;g) \qquad \vartriangle\text{-FUSION}$$

Law UNIQ now asserts that $outl, outr$ are jointly monic. (A morphism f is *monic* if $x\,;f = y\,;f$ implies $x = y$, for all x, y.)

Application As a first application we show that inl is monic (and by symmetry inr too, and dually each of $outl$ and $outr$ is epic):

$$\begin{array}{ll} & x = y \\ \equiv & \quad \text{aiming at } ``\ ;inl" \text{ after } x \text{ and } y\text{, use } \triangledown\text{-SELF}[f := id] \\ & x\,;inl\,;id \triangledown g = y\,;inl\,;id \triangledown g \\ \Leftarrow & \quad \text{Leibniz} \end{array}$$

$$x\,;inl = y\,;inl$$

as desired. The choice for g is immaterial; id certainly does the job.

As a second application we show that $\triangledown$ and $\triangle$ *abide* (expressed by the next line):

$$\begin{array}{ll} & (f \triangledown g) \triangle (h \triangledown j) = (f \triangle h) \triangledown (g \triangle j) \\ \equiv & \quad \triangledown\text{-CHARN}\,[x, f, g := \text{lhs},\ f \triangle h,\ g \triangle j] \\ & inl\,;(f \triangledown g) \triangle (h \triangledown j) = f \triangle h \ \ \wedge \ \ inr\,;(f \triangledown g) \triangle (h \triangledown j) = g \triangle j \\ \equiv & \quad \triangle\text{-FUSION (at two places)} \\ & (inl\,; f \triangledown g) \triangle (inl\,; h \triangledown j) = f \triangle h \ \ \wedge \ \ (inr\,; f \triangledown g) \triangle (inr\,; h \triangledown j) = g \triangle j \\ \equiv & \quad \triangledown\text{-SELF (at four places)} \\ & f \triangle h = f \triangle h \ \ \wedge \ \ g \triangle j = g \triangle j \\ \equiv & \quad \text{equality} \\ & \mathsf{true}. \end{array}$$

For later use we define, for $f : a \to b$ and $g : c \to d$,

$$\begin{array}{lcll} f + g & = & (f\,; inl) \triangledown (g\,; inr) & : \ a + c \to b + d \\ f \times g & = & (outl\,; f) \triangle (outr\,; g) & : \ a \times c \to b \times d\,. \end{array}$$

The reader may wish to prove that $+$ and $\times$ are bifunctors: $id+id = id$ and $f+g;h+j = (f\,;h) + (g\,;j)$, and similarly for $\times$.

4 Algebras

Let F be a functor from $\mathbf{C}$ to $\mathbf{C}$. Recall from Section 2 that an F-algebra is just a morphism $\phi : Fa \to a$ for some a, called the carrier of ϕ, and a homomorphism from ϕ to ψ is a morphism f satisfying $\phi\,; f = Ff\,; \psi$. It follows that $f : \text{carrier}\,\phi \to \text{carrier}\,\psi$. We abbreviate $\to_{\text{Alg}(F)}$ to just $\to_F$. To motivate the terminology, consider the following two cases. (An extensive motivation is given by Fokkinga and Meijer [10].)

A single binary operation $\phi : a \times a \to a$ is an F-algebra, where functor F is defined by $Fx = x \times x$. Then '$f : \phi \to_F \psi$' means $\phi ; f = f \times f ; \psi$, saying that f commutes with the operations. Now consider two algebras (or n-ary operations) of different type but with the same carrier, say $\phi_i : F_i a \to a$ for $i = 0, 1$. We can then form $\phi = \phi_0 \triangledown \phi_1 : F_0 a + F_1 a \to a$; this is an F-algebra, where functor F is defined by $Fx = F_0 x + F_1 x$. From the composite ϕ we can retrieve the constituent operations by $\phi_0 = inl\,; \phi$ and $\phi_1 = inr\,; \phi$. Statement $f : \phi_0 \triangledown \phi_1 \to_{F_0+F_1} \psi_0 \triangledown \psi_1$ boils down to the two statements $f : \phi_i \to_{F_i} \psi_i$ for $i = 0, 1$. Thus, classical single-sorted algebras can be modeled as F-algebras. In another paper [8] I've shown how to deal with equations (like associativity) in this frame-work. Many-sortedness and general 'datatypes' require only a slight generalisation [9, 8].

Let α be initial in $\mathrm{Alg}(F)$ (supposing it exists). We fix this α throughout what follows, and we write $(\!|\phi|\!)$ for $(\!|\alpha \to \phi|\!)_{\mathrm{Alg}(F)}$, and call it a *catamorphism*. Then the laws for α and $(\!|\ |\!)$ work out as follows.

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

Let a be the carrier of α. Below we shall show that α is an isomorphism: $\alpha : Fa \to a$, the inverse of which we denote $\alpha^{\cup}$. We might call α a "constructor", since in **Set** each element of a can be obtained as an outcome of α for precisely one argument, called the "constituents" of the element. The inverse $\alpha^{\cup}$ is then a "destructor"; it maps each element of a into the constituents. Using $\alpha^{\cup}$, the premise of cata-CHARN may be reformulated as $x = \alpha^{\cup}\,;Fx\,;\phi$. Thus cata-CHARN says that the "inductive definition" $x = \alpha^{\cup}\,;Fx\,;\phi$ has a unique solution for x. (If α were not initial, such an equation might have no or several solutions.) Similarly, cata-UNIQ says that if two morphisms x and y both satisfy the same "inductive pattern", namely $x = \alpha^{\cup}\,;Fx\,;\phi$ and $y = \alpha^{\cup}\,;Fy\,;\phi$, then they are the same. One sees that cata-UNIQ captures, in a sense, induction. Law cata-FUSION may also be read as giving a sufficient condition on x and ϕ in order that the composite $(\!|\phi|\!)\,;x$ can be expressed in the form $(\!|...|\!)$. Much more explanation is given by Fokkinga and Meijer [10].

Application As an example calculation let us show that the initial F-algebra α is — up to isomorphism— a fixed point of F, i.e., $F\alpha \simeq \alpha$ in $\mathrm{Alg}(F)$. This requires us to establish a pair x, y of morphisms in $\mathrm{Alg}(F)$ (F-algebra homomorphisms in $\mathbf{C}$),

$$\begin{array}{lcl}
x & : & F\alpha \to_F \alpha \\
y & : & \alpha \to_F F\alpha\,,
\end{array}$$

that are each others inverse. Since initiality of α in $\mathrm{Alg}(F)$ is given, we know by the proof of the uniqueness of initial objects that the only choice for y is $(\!|F\alpha|\!)$. By unfolding the requirement for $\mathbf{x}$ we immediately find that the only choice for x is α. It remains to show that these choices are each others inverse indeed. For this we argue:

$$\begin{array}{ll}
 & (\!|F\alpha|\!)\,;\alpha = id \\
\equiv & \quad \text{cata-ID} \\
 & (\!|F\alpha|\!)\,;\alpha = (\!|\alpha|\!) \\
\Leftarrow & \quad \text{cata-FUSION} \\
 & F\alpha\,;\alpha = F\alpha\,;\alpha \\
\equiv & \quad \text{equality}
\end{array}$$

true.

So $(\!|F\alpha|\!)$ is a pre-inverse of α. It is a post-inverse too:

$$
\begin{array}{ll}
 & \alpha\,;(\!|F\alpha|\!) \\
= & \quad \text{cata-SELF} \\
 & F(\!|F\alpha|\!)\,;F\alpha \\
= & \quad \text{functor, above: } (\!|F\alpha|\!) \text{ is pre-inverse of } \alpha \\
 & Fid \\
= & \quad \text{functor} \\
 & id.
\end{array}
$$

Since $\mathrm{Alg}(F)$ is built upon $\mathbf{C}$, an immediate corollary of the isomorphism $\alpha : F\alpha \simeq \alpha$ in $\mathrm{Alg}(F)$ is the isomorphism $\alpha : F(\mathrm{carrier}\,\alpha) \simeq \mathrm{carrier}\,\alpha$ in $\mathbf{C}$.

5 Coequalisers and Kernel pairs

Let $\mathbf{C}$ be **Set**. Each parallel pair (f,g) with common target a represents a relation $R_{f,g}$ on a, namely $R_{f,g} = \{(f\,z,\,g\,z)|\ z \text{ in the common source of } f,g\} \subseteq a \times a$. Also, each morphism p with source a represents an equivalence relation R_p on a, namely $R_p = \{(x,y)|\ p\,x = p\,y\} \subseteq a \times a$. Observe that $R_{f,g} \subseteq R_p$ if and only if $f\,;p = g\,;p$.

A p for which R_p is the least equivalence relation including relation $R_{f,g}$, is categorically characterised as coequaliser of (f,g). A pair (f,g) for which $R_{f,g}$ is the greatest relation included in an equivalence relation R_p is categorically characterised as kernel pair of p. We shall present their properties in a way suitable for calculation.

Coequalisers

By definition, a *coequaliser* of a parallel pair (f,g) is an initial object in $\bigvee(f\|g)$. Let p be a coequaliser of (f,g), supposing one exists. We write $p\backslash_{f,g}q$ or simply $p\backslash q$ instead of $(\!|p \multimap q|\!)_{\bigvee(f\|g)}$ since, as we shall explain, the fraction notation better suggests the calculational properties. Then the laws for p and $\backslash$ work out as follows.

$$
\begin{array}{lllll}
 & p\,;x = q & \equiv & x = p\backslash q & \backslash\text{-CHARN} \\
 & p\,;p\backslash q = q & & & \backslash\text{-SELF} \\
 & id = p\backslash p & & & \backslash\text{-ID} \\
 & p\,;x = q \ \wedge\ p\,;y = q & \Rightarrow & x = y & \backslash\text{-UNIQ} \\
\text{i.e.,} & p\,;x = p\,;y & \Rightarrow & x = y & (p \text{ epic}) \\
 & q\,;x = r & \Rightarrow & p\backslash q\,;x = p\backslash r & \backslash\text{-FUSION} \\
\text{i.e.,} & p\backslash q\,;x = p\backslash(q\,;x) & & &
\end{array}
$$

In the laws \-CHARN, \-SELF and \-FUSION we have omitted the premise that q is an object in $\mathsf{V}(f\|g)$; this is rather harmless if we agree that the notation $\ldots\backslash q$ requires a separate proof of the well-formedness condition $f\,;q = g\,;q$. Notice also how \-FUSION simplifies to an unconditional fusion law. Similarly law \-UNIQ simplifies to the assertion that a coequaliser is epic.

Now that we have presented the laws the choice of notation may be evident: the usual manipulation of *cancelling adjacent factors in the denominator and nominator* is valid when composition is interpreted as multiplication and $\backslash$ is interpreted as a fraction. (See also law \-COMPOSE below.) This may also help you to remember that there is only post-fusion here; the equation $x\,;p\backslash q = (x\,;p)\backslash q$ is not meaningful and not valid in general.

Additional laws As the reader may expect, the following law is valid too:

$$p\backslash q\,;\, q\backslash r = p\backslash r \qquad \backslash\text{-COMPOSE}$$

One can prove this in various ways; here is one way:

$$\begin{array}{ll} & p\backslash q\,;\, q\backslash r \\ = & \quad \backslash\text{-FUSION} \\ & p\backslash(q\,;q\backslash r) \\ = & \quad \backslash\text{-SELF} \\ & p\backslash r. \end{array}$$

The interesting aspect here is that the suppressed subscripts to $\backslash$ may differ: e.g., $p\backslash_{f,g}q$ and $q\backslash_{h,j}r$, and q is not necessarily a coequaliser of f,g. Rephrased in the standard notation, law \-COMPOSE reads:

$$(\!|a \to b|\!)_{\mathcal{A}}\,;(\!|b \to c|\!)_{\mathcal{B}} = (\!|a \to c|\!)_{\mathcal{A}} \qquad \text{COMPOSE}$$

where $\mathcal{A}$ and $\mathcal{B}$ are full subcategories of some category $\mathcal{C}(\mathbf{C})$ and objects b,c are in both $\mathcal{A}$ and $\mathcal{B}$; in our case $\mathcal{A} = \mathsf{V}(f\|g)$, $\mathcal{B} = \mathsf{V}(h\|j)$, and $\mathcal{C}(\mathbf{C}) = \mathsf{V}(d)$ where d is the common target of f,g,h,j. Then the proof runs as follows.

$$\begin{array}{ll} & (\!|a \to b|\!)_{\mathcal{A}}\,;(\!|b \to c|\!)_{\mathcal{B}} = (\!|a \to c|\!)_{\mathcal{A}} \\ \Leftarrow & \quad \text{FUSION} \\ & (\!|b \to c|\!)_{\mathcal{B}} : b \to_{\mathcal{A}} c \\ \equiv & \quad \text{both } \mathcal{A} \text{ and } \mathcal{B} \text{ are full subcategories of } \mathcal{C}(\mathbf{C}), \\ & \quad \text{each containing both } b \text{ and } c \\ & (\!|b \to c|\!)_{\mathcal{B}} : b \to_{\mathcal{B}} c \\ \equiv & \quad \text{SELF} \\ & \mathsf{true}. \end{array}$$

Another law that we shall use below has to do with functors.

$$\text{if } Fp\backslash\ldots \text{ is well-defined, then} \qquad F(p\backslash q) = Fp\backslash Fq \qquad \backslash\text{-FCTR}$$

Clearly, the premise of \-FCTR is met when F preserves coequalisers. For the proof we argue

$$\begin{array}{ll} & F(p\backslash q) = Fp\backslash Fq \\ \equiv & \quad \backslash\text{-CHARN} \\ & Fp\,;F(p\backslash q) = Fq \\ \equiv & \quad \text{functor} \\ & F(p\,;p\backslash q) = Fq \\ \equiv & \quad \backslash\text{-SELF} \\ & \textsf{true}. \end{array}$$

Kernel pairs

By definition, a *kernel pair* of a morphism p is a final object in $\bigwedge(p \_\!\rfloor p)$.[1] Let (f,g) be a kernel pair of p, supposing it exists. This time we use the notation $(d,e)/_p(f,g)$ or simply $(d,e)/(f,g)$ instead of $[\![d,e \to f,g]\!]_{\bigwedge(p \_\!\rfloor p)}$ (for which the well-formedness condition reads $d\,;p = e\,;p$). Then the laws for (f,g) and $/$ work out as follows.

$$\begin{array}{lll} & d = x\,;f \;\wedge\; e = x\,;g & \equiv \quad x = (d,e)/(f,g) \qquad /\text{-CHARN} \\ & d = (d,e)/(f,g)\,;f \;\wedge\; e = (d,e)/(f,g)\,;g & \qquad\qquad\qquad\quad /\text{-SELF} \\ & id = (f,g)/(f,g) & \qquad\qquad\qquad\quad /\text{-ID} \\ & \left.\begin{array}{l} d = x\,;f \;\wedge\; e = x\,;g \\ d = y\,;f \;\wedge\; e = y\,;g \end{array}\right\} & \Rightarrow \quad x = y \qquad /\text{-UNIQ} \\ \text{i.e.,} & x\,;f = y\,;f \;\wedge\; x\,;g = y\,;g & \Rightarrow \quad x = y \quad ((f,g) \text{ jointly monic}) \\ & x\,;(d,e)/(f,g) = (x\,;d,\; x\,;e)/(f,g) & \qquad\qquad\qquad\quad /\text{-FUSION} \\ & (d,e)/(f,g)\,;(f,g)/(h,j) = (d,e)/(h,j) & \qquad\qquad\qquad\quad /\text{-COMPOSE} \\ & \text{if } \ldots/F(f,g) \text{ is well-defined, then } F((d,e)/(f,g)) = F(d,e)/F(f,g) & \qquad\qquad\qquad\quad /\text{-FCTR} \end{array}$$

Notice that there is pre-fusion only. Due to the presence of so many pairs the notation is quite cumbersome, but we refrain from simplifying it here.

Application As an example of the use of the laws we prove that the coequaliser and kernel pair form an adjunction. Specifically, let C denote a mapping that sends each parallel pair with common target a to some coequaliser of it, and similarly let K send

[1] Don't bother about the definition of $\mathcal{C} = \bigwedge(p \_\!\rfloor p)$. All that matters is the laws that follow. ($\mathcal{C}$ is the category of cones for the diagram $\_\!\rfloor$ with both legs being p, so that a kernel pair of p is just a pullback of p,p.)

each morphism with source a to some kernel pair of it. We extend these mapping to functors $C : \bigwedge(a,a) \to \bigvee(a)$ and $K : \bigvee(a) \to \bigwedge(a,a)$, by defining

$$\begin{array}{lcll} Cx & = & C(d,e)\backslash C(f,g) & : \quad C(d,e) \to_{\bigvee(a)} C(f,g) \quad \text{for } x : (d,e) \to_{\bigwedge(a,a)} (f,g) \\ Ku & = & Kp/Kq & : \quad Kp \to_{\bigwedge(a,a)} Kq \quad \text{for } u : p \to_{\bigvee(a)} q \end{array}$$

We shall establish natural transformations $\epsilon : CK \dot{\to} I$ and $\eta : I \dot{\to} KC$ such that $\eta K \,;\, K\epsilon = idK$ and $C\eta \,;\, \epsilon C = idC$. Take $\epsilon q = CKq\backslash q : CKq \to_{\bigvee(a)} q$ for all q in $\bigvee(a)$. The naturality of ϵ is shown as follows. For arbitrary $u : q \to_{\bigvee(a)} r$,

$$\begin{array}{ll} & CKu \,;\, \epsilon r \\ = & \quad \text{definition } C,\ K \text{ and } \epsilon, \text{ noting that } u : q \to_{\bigvee(a)} r \\ & CKq\backslash CKr \,;\, CKr\backslash r \\ = & \quad \backslash\text{-COMPOSE} \\ & CKq\backslash r \\ = & \quad \text{equation } \text{“}u : q \to_{\bigvee(a)} r\text{”} \\ & CKq\backslash(q \,;\, u) \\ = & \quad \backslash\text{-FUSION} \\ & CKq\backslash q \,;\, u \\ = & \quad \text{definition } \epsilon \text{ and } I \\ & \epsilon q \,;\, Iu \end{array}$$

as desired. Further we take $\eta(d,e) = (d,e)/KC(d,e) : (d,e) \to_{\bigwedge(a,a)} KC(d,e)$. We omit the proof that η is natural; this is quite similar (but not categorically dual) to the naturality of ϵ. Next we show that $\eta K \,;\, K\epsilon = idK$. Let q be arbitrary, then

$$\begin{array}{ll} & (\eta K \,;\, K\epsilon)q \\ = & \quad \text{definitions of } \eta,\ \epsilon, \text{ and composition of natural transformations} \\ & Kq/KCKq \,;\, K(CKq\backslash q) \\ = & \quad \text{definition } K \text{ (see above with } u,p,q := q\backslash CKq, q, CKq) \\ & K(q\backslash CKq) \,;\, K(CKq\backslash q) \\ = & \quad \text{functor, } \backslash\text{-COMPOSE} \\ & K(q\backslash q) \\ = & \quad \backslash\text{-ID} \\ & K(id_q) \qquad \text{notice that } id_q(: q \to_{\bigvee(a)} q) \text{ is } id_{\mathrm{tgt}q}(: \mathrm{tgt}q \to \mathrm{tgt}q) \\ = & \quad \text{functor} \\ & idKq. \end{array}$$

The proof of $C\eta \,;\, \epsilon C = idC$ is again quite similar to the above one.

6 Colimits

An initial object is just a colimit of the empty diagram, and conversely, a colimit of a diagram is just an initial object in the category of cocones over that diagram. Let us therefore present the algebraic properties of colimits.

A *diagram* D is a collection of morphisms. The sources and targets of members of D are collectively called its objects. Category $\bigvee D$, built upon $\mathbf{C}$, is defined as follows. An object in $\bigvee D$, called *cocone* for D, is a collection γ of morphisms $\gamma_a : a \to c$ (one for each object a in D) where c is some common target (called the target of γ), satisfying

$$\forall(f : a \to b \text{ in } D :: \quad \gamma_a = f ; \gamma_b) .$$

We write $\gamma = \{a \text{ object in } D :: \ \gamma_a\}$, or simply $\gamma = \{a :: \ \gamma_a\}$ when D is understood. (A cocone must have a distinguished target even if it is empty.) Let γ and δ be cocones for D; then a morphism from γ to δ in $\bigvee D$ is a morphism x in $\mathbf{C}$ satisfying

$$\forall(a \text{ in } D :: \quad \gamma_a ; x = \delta_a) .$$

By definition, a *colimit for D in $\mathbf{C}$* is an initial object in $\bigvee D$. Let γ be a colimit for D, supposing it exists. We write $\gamma\backslash_D\delta$ or simply $\gamma\backslash\delta$, instead of $(\!|\gamma \multimap \delta|\!)_{\bigvee D}$. Then the laws for γ and $\backslash$ work out as follows (a ranges over the objects of D).

$\forall(a :: \ \gamma_a ; x = \delta_a)$	$\equiv$	$x = \gamma\backslash\delta$	\-CHARN
$\forall(a :: \ \gamma_a ; \gamma\backslash\delta = \delta_a)$			\-SELF
$id = \gamma\backslash\gamma$			\-ID
$\forall(a :: \ \gamma_a ; x = \gamma_a ; y)$	$\Rightarrow$	$x = y$ (γ jointly epic)	\-UNIQ
$\gamma\backslash\delta ; x = \gamma\backslash\{a :: \ \delta_a ; x\}$			\-FUSION
$\gamma\backslash\delta ; \delta\backslash\epsilon = \gamma\backslash\epsilon$			\-COMPOSE
if $F\gamma\backslash\ldots$ defined, then		$F(\gamma\backslash\delta) = F\gamma\backslash F\delta$	\-FCTR

for each D-cocone δ, ϵ (δ being a colimit in law \-COMPOSE).

Improved description In view of the explicit quantifications the above laws for colimits are not very suited for algebraic calculation. We can avoid a lot of explicit quantifications by treating a cocone as a family of functions, and defining for example $\gamma ; x = \delta$ to mean $\forall(a :: \ \gamma_a ; x = \delta_a)$. It turns out that we can perform this trick in a categorical fashion by using natural transformations, which are families of morphisms indeed. Several (not all) manipulations on the subscripts can then be phrased as well-known manipulations with natural transformations as a whole. So let us redesign the definitions. (I've got the suggestion from Jaap van der Woude; Lambek and Scott [18] use the following formulation too.)

As regards the property of being a cocone we can assume without loss of generality that a diagram is a subcategory: just add all the compositions of composable arrows, and

all the appropriate identities. Going one step further we can consider the subcategory to be the image under a functor $D : \mathbf{D} \to \mathbf{C}$, where $\mathbf{D}$ is a category that gives the shape of the diagram. So we define: a *diagram* in $\mathbf{C}$ is a functor $D : \mathbf{D} \to \mathbf{C}$, for some category $\mathbf{D}$, called the *shape* of the diagram. Now we define category $\bigvee D$ as follows. An object in $\bigvee D$, again called *cocone* for D, is a natural transformation $\gamma : D \dot{\to} \underline{c}$ for some object c in $\mathbf{C}$ ($\underline{c}$ is the constant functor mapping each object a into c, and each morphism x into the identity $id_c : c \to c$). Indeed, by naturality we have for each $Df : Da \to Db$ in the 'diagram' $D\mathbf{D}$ in $\mathbf{C}$

$$\begin{array}{lcll} \gamma a \,;\, \underline{c}f & = & Df \,;\, \gamma b & : \quad Da \to \underline{c}a \qquad \text{that is,} \\ \gamma a & = & Df \,;\, \gamma b & : \quad Da \to c \end{array}$$

which expresses the required "commutativity of the triangle." Let γ and δ be cocones for D; then a morphism from γ to δ in $\bigvee D$ is a morphism x in $\mathbf{C}$ satisfying $\gamma \,;\, x = \delta$ (the compostion of γ and x is defined below). Again, a *colimit for* D is an initial object in $\bigvee D$.

Since cocones are natural transformations, we have the following standard transformations available. For $\gamma : D \dot{\to} \underline{c}$ and $\delta : D \dot{\to} \underline{d}$:

- for each $x : c \to d$,
  $\gamma \,;\, x = \lambda(a :: \ \gamma a \,;\, x) : D \dot{\to} \underline{d}$ is a cocone for D again.
- for each functor $F : \mathbf{C} \to \mathbf{C}$,
  $F\gamma = \lambda(a :: \ F(\gamma a)) : FD \dot{\to} F\underline{c}$ is a cocone for FD (note that $F\underline{c} = \underline{Fc}$).
  If in addition F preserves colimits, then $F\gamma$ is a colimit for FD if γ is so for D.
  Since $(F\gamma)a = F(\gamma a)$ we omit the parentheses.
- for each functor $S : \mathbf{D} \to \mathbf{D}$,
  $\gamma S = \lambda(a :: \ \gamma(Sa)) : DS \dot{\to} \underline{c}$ is a cocone for DS (note that $\underline{c}S = \underline{c}$).
  If S transforms the shape somewhat, γS is the transformed cocone.
  Since $\gamma(Sa) = (\gamma S)a$ we omit the parentheses.

Let γ be a colimit for D. Then the laws for γ and $\backslash$ work out as follows.

$$\begin{array}{lcll} \gamma \,;\, x = \delta & \equiv & x = \gamma\backslash\delta & \backslash\text{-CHARN} \\ \gamma \,;\, \gamma\backslash\delta = \delta & & & \backslash\text{-SELF} \\ \gamma\backslash\gamma = id & & & \backslash\text{-ID} \\ \gamma \,;\, x = \gamma \,;\, y & \Rightarrow & x = y \quad (\gamma \text{ jointly epic}) & \backslash\text{-UNIQ} \\ \gamma\backslash\delta \,;\, x = \gamma\backslash(\delta \,;\, x) & & & \backslash\text{-FUSION} \\ \gamma\backslash\delta \,;\, \delta\backslash\epsilon = \gamma\backslash\epsilon & & & \backslash\text{-COMPOSE} \\ \text{if } F\gamma\backslash\ldots \text{ defined, then} & & F(\gamma\backslash\delta) = F\gamma\backslash F\delta & \backslash\text{-FCTR} \end{array}$$

for each D-cocone δ, ϵ (δ being a colimit in law $\backslash$-COMPOSE). Notice also that

$$\gamma S \,;\, \gamma\backslash\delta \ = \ (\gamma \,;\, \gamma\backslash\delta)S \ = \ \delta S,$$

but in general $\gamma\backslash\delta \neq \gamma S\backslash\delta S$ since γS need not be a colimit and therefore the latter right-hand side is not well-defined.

Application We present the well-known construction of an initial F-algebra. Our interest is solely in the algebraic, calculational style of various subproofs.

Given endofunctor F we wish to construct an F-algebra, $\alpha : Fa \to a$ say, that is initial in $\mathrm{Alg}(F)$. Forgoing initiality for the time being, we derive a construction of an $\alpha : Fa \to a$ as follows.

$$
\begin{array}{ll}
 & \alpha : Fa \to a \\
\Leftarrow & \quad \text{definition isomorphism} \qquad \text{(a)} \\
 & \alpha : Fa \simeq a \\
\Leftarrow & \quad \text{definition cocone morphism (taking } a = \mathrm{tgt}\gamma = \mathrm{tgt}\gamma S\text{)} \qquad \text{(b)} \\
 & \alpha : F\gamma \simeq \gamma S \text{ in } \bigvee(FD) \quad \wedge \quad FD = DS \\
\equiv & \quad F\gamma \text{ is colimit for } FD \text{ (taking } \alpha = F\gamma \backslash \gamma S\text{)} \qquad \text{(c)} \\
 & \gamma S \text{ is colimit for } DS \quad \wedge \quad FD = DS.
\end{array}
$$

Step (a) is motivated by the wish that α be initial in $\mathrm{Alg}(F)$, and so α will be an isomorphism (see the section on algebras); in other words, in view of the required initiality the step is *no* weakening. In step (b) we merely decide that α, a come from a (co)limit construction; this is true for virtually all categorical constructions. In order that the step is valid we have to define a diagram D, a D-cocone γ (a colimit say, which we *assume* to exist), and we have to define an endofunctor S on $\mathrm{src}D$ (that transforms D-cocone γ into a DS $(= FD)$-cocone γS). In step (c) the claim '$F\gamma$ is colimit for FD' follows from the *assumption* that F preserves colimits. The definition $\alpha = F\gamma \backslash \gamma S$ is *forced* by (the proof of) the uniqueness of initial objects. (It is indeed very easy to verify that $F\gamma \backslash \gamma S$ and $\gamma S \backslash F\gamma$ are each others inverse.) We shall now complete the construction in the following three parts.

1. Construction of D, S such that $FD = DS$.
2. Proof of 'γS is colimit for DS' where γ is a colimit for D.
3. Proof of 'α is initial in $\mathrm{Alg}(F)$' where $\alpha = F\gamma \backslash \gamma S$.

Part 1 (Construction of D, S such that $FD = DS$.) The requirement $FD = DS$ says that FD is a 'subdiagram' of D. This is easily achieved by making D a *chain* of iterated F applications, as follows.

Let ω be the category with objects $0, 1, 2, \ldots$ and a unique arrow from i to j (denoted $i{\leq}j$) for every $i \leq j$. So ω is the shape of a chain. The zero and successor functors $0, S : \omega \to \omega$ are defined by $0(i{\leq}j) = 0{\leq}0$ and $S(i{\leq}j) = (i{+}1){\leq}(j{+}1)$.

Let $\mathbf{0}$ be an initial object in $\mathbf{C}$. Define the diagram $D : \omega \to \mathbf{C}$ by $D(i{\leq}j) = F^i(\!|F^{j-i}\mathbf{0}|\!)$ (where $(\!|\_|\!)$ abbreviates $(\!|\mathbf{0} \leftarrow \_|\!)_{\mathbf{C}}$). It is quite easy to show that D is a functor, i.e., $D(i{\leq}j\,;\, j{\leq}k) = D(i{\leq}j)\,;\, D(j{\leq}k)$. It is also immediate that $FD = DS$, since

$$FD(i{\leq}j) \;=\; FF^i(\!|F^{j-i}\mathbf{0}|\!) \;=\; F^{i+1}(\!|F^{(j+1)-(i+1)}\mathbf{0}|\!) \;=\; D((i{+}1){\leq}(j{+}1)) \;=\; DS(i{\leq}j).$$

Part 2 (Proof of 'γS is colimit for DS' where γ is a colimit for D.) Our task is to construct a morphism $(\!|\gamma S \dashv \delta|\!)_{\forall(DS)}$ such that

$$(\spadesuit) \quad \gamma S\,;x = \delta \quad \equiv \quad x = (\!|\gamma S \dashv \delta|\!)_{\forall(DS)}$$

for each $\delta : DS \nrightarrow \underline{d}$ (an arbitrary cocone). Our guess is that $\gamma\backslash(\delta', \delta)$ may be chosen for $(\!|\gamma S \dashv \delta|\!)_{\forall(DS)}$, where $(\delta', \delta) : D \nrightarrow \underline{d}$ is defined by

$$\begin{array}{lllllll} (\delta',\delta)0 & = & \delta' & : & D0 \nrightarrow \underline{d}0 & \approx & \mathbf{0} \to d \\ (\delta',\delta)S & = & \delta & : & DS \nrightarrow \underline{d}S & = & DS \nrightarrow \underline{d} \end{array}$$

for some suitably chosen δ'. We prove the desired equivalence $(\spadesuit)$, deriving along the way a requirement for δ'.

$$\begin{array}{ll} & x = \gamma\backslash(\delta', \delta) \\ \equiv & \quad \backslash\text{-CHARN} \\ & \gamma\,;x = (\delta', \delta) \\ \equiv & \quad \text{property natural transformations, } \omega \text{ and } 0, S \\ & (\gamma\,;x)0 = (\delta', \delta)0 \quad \wedge \quad (\gamma\,;x)S = (\delta', \delta)S \\ \equiv & \quad \text{calculus for natural transformations, defn } (\delta', \delta) \\ & \gamma 0\,;x = \delta' \quad \wedge \quad \gamma S\,;x = \delta \\ \equiv & \quad \textbf{define } \delta' \textbf{ below} \text{ such that } \gamma 0\,;x = \delta' \text{ for each } x \text{ (using } \gamma S\,;x = \delta\,) \qquad (*) \\ & \gamma S\,;x = \delta. \end{array}$$

In order to define δ' satisfying the requirement derived at step $(*)$, we calculate

$$\begin{array}{ll} & \gamma 0\,;x \\ = & \quad \{\text{anticipating next steps, introduce an identity}\} \\ & \gamma 0\,;\underline{c}(0{\leq}1)\,;x \\ = & \quad \text{naturality } \gamma \text{ (“commutativity of the triangle”)} \\ & D(0{\leq}1)\,;\gamma 1\,;x \\ = & \quad \text{using } \gamma S\,;x = \delta \\ & D(0{\leq}1)\,;\delta 0 \end{array}$$

so that we can fulfill the requirement $\gamma 0\,;x = \delta'$ by defining $\delta' = D(0{\leq}1)\,;\delta 0$.

Part 3 (Proof of 'α is initial in $\mathrm{Alg}(F)$' where $\alpha = F\gamma\backslash\gamma S$.) Let $\phi : Fa \to a$ be arbitrary. We have to construct a morphism $(\!|\alpha \dashv \phi|\!)_F : c \to a$ in $\mathbf{C}$ such that

$$(\clubsuit) \quad F\gamma\backslash\gamma S\,;x = Fx\,;\phi \quad \equiv \quad x = (\!|\alpha \dashv \phi|\!)_F\,.$$

Our guess is that the required morphism $(\!|\alpha \to \phi|\!)_F$ can be written as $\gamma\backslash\delta$ for some suitably chosen D-cocone δ. Based on type considerations one may derive a definition of δ; we just guess it.

$$\begin{array}{lcl} \delta 0 & = & (\!|a|\!)_{\mathbf{C}} \\ \delta S & = & F\delta\,;\phi\,. \end{array}$$

It remains to show that $\gamma\backslash\delta$ for $(\!|\alpha \to \phi|\!)_F$ fulfills the required equivalence (♣). First we show the $\Leftarrow$ part:

$$\begin{array}{ll} & F\gamma\backslash\gamma S\,;\gamma\backslash\delta \\ = & \quad \backslash\text{-FUSION} \\ & F\gamma\backslash(\gamma S\,;\gamma\backslash\delta) \\ = & \quad \text{observed just after the colimit laws} \\ & F\gamma\backslash\delta S \\ = & \quad \text{definition } \delta S = F\delta\,;\phi \\ & F\gamma\backslash(F\delta\,;\phi) \\ = & \quad \backslash\text{-FUSION} \\ & F\gamma\backslash F\delta\,;\phi \\ = & \quad \text{functor } F \text{ preserves colimits} \\ & F(\gamma\backslash\delta)\,;\phi \end{array}$$

as desired. Next we show the $\Rightarrow$ part.

$$\begin{array}{ll} & x = \gamma\backslash\delta \\ \equiv & \quad \backslash\text{-CHARN} \\ & \gamma\,;x = \delta \\ \Leftarrow & \quad \text{induction principle} \\ & (\gamma\,;x)0 = \delta 0 \;\;\wedge\;\; \forall(n ::\;\; (\gamma\,;x)n = \delta n \;\;\Rightarrow\;\; (\gamma\,;x)Sn = \delta Sn) \\ \equiv & \quad \text{proved below in (i) and (ii)} \\ & \mathsf{true}. \end{array}$$

For (i) we calculate

$$\begin{array}{ll} & \gamma 0\,;x \\ = & \quad \text{CHARN, using } \gamma 0 : \mathbf{0} \to c \\ & (\!|c|\!)_{\mathbf{C}}\,;x \\ = & \quad \text{FUSION, using } x : c \to a \\ & (\!|a|\!)_{\mathbf{C}} \\ = & \quad \text{definition } \delta 0 \\ & \delta 0 \end{array}$$

as desired. And for (ii) we calculate for arbitrary n, assuming $(\gamma\,;x)n = \delta n$,

$$
\begin{array}{ll}
 & (\gamma S\,;x)n \\
= & \quad \backslash\text{-SELF (aiming at using the premise of } (\clubsuit)\,) \\
 & (F\gamma\,;F\gamma\backslash\gamma S\,;x)n \\
= & \quad \text{premise, see } (\clubsuit) \\
 & (F\gamma\,;Fx\,;\phi)n \\
= & \quad \text{functor} \\
 & (F(\gamma\,;x)\,;\phi)n \\
= & \quad \text{hypothesis } (\gamma\,;x)n = \delta n \\
 & (F\delta\,;\phi)n \\
= & \quad \text{definition } \delta S \\
 & (\delta S)n
\end{array}
$$

as desired. This completes the entire construction and proof.

Acknowledgement This note is heavily influenced by what other people have taught me in the past two years. In particular Roland Backhouse, Grant Malcolm, Lambert Meertens and Jaap van der Woude may recognise their ideas and methodological and notational suggestions. An anonymous CSN referee has read the paper very critically and suggested many corrections.

References

[1] M. Arbib and E. Manes. *Arrows, Structures and Functors — The Categorical Imperative.* Academic Press, 1975.

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

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

[4] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, pages 3–42. Springer Verlag, 1987. Also Technical Monograph PRG–56, Oxford University, Computing Laboratory, Programming Research Group, October 1986.

[5] R.S. Bird. Lecture notes on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science.* International Summer School directed by F.L. Bauer [et al.], Springer Verlag, 1989. NATO Advanced Science Institute Series (Series F: Computer and System Sciences Vol. 55).

[6] G. Cousineau, P.L. Curien, and M. Mauny. The categorical abstract machine. In *Functional Programming Languages and Computer Architecture*, volume 201 of *Lect. Notes in Comp. Sc.* Springer Verlag, 1985.

[7] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Springer Verlag, 1990.

[8] M.M. Fokkinga. Datatype laws without signatures. Technical Report CS-R9133, CWI, Amsterdam, Amsterdam, May 1991.

[9] M.M. Fokkinga. Initial many-sorted algebras. CWI, Amsterdam, May 1991.

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

[11] A.J.M. van Gasteren. *On the shape of Mathematical Arguments.* PhD thesis, Technical University of Eindhoven, 1988.

[12] R. Goldblatt. *Topoi — the Categorical Analysis of Logic*, volume 98 of *Studies in Logic and the Foundations of mathematics.* North-Holland, 1979.

[13] J. Gray and A. Scedrov, editors. *Categories in Computer Science and Logic*, volume 92 of *Contempory mathematics.* 1989.

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

[15] T. Hagino. A typed lambda calculus with categorical type constructors. In D.H. Pitt, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, number 283 in Lect. Notes in Comp. Sc., pages 140–157. Springer Verlag, 1987.

[16] C.A.R. Hoare. Notes on an Approach to Category Theory for Computer Scientists. In M. Broy, editor, *Constructive Methods in Computing Science*, pages 245–305. International Summer School directed by F.L. Bauer [et al.], Springer Verlag, 1989. NATO Advanced Science Institute Series (Series F: Computer and System Sciences Vol. 55).

[17] J. Hughes. Why functional programming matters. In D.A. Turner, editor, *Research Topics in Functional Programming*, pages 17–42. Addison-Wesley, 1990.

[18] J. Lambek and P.J. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge Studies in advanced mathematics*. Cambridge University Press, 1986.

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

[20] S. MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, 1971.

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

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

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

[24] 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.

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

[26] D. Pitt, A. Abramsky, A. Poigné, and D. Rydeheard, editors. *Category Theory and Computer Science*, volume 240 of *Lect. Notes in Comp. Sc.* Springer Verlag, 1986.

[27] D. Pitt, A. Poigné, and D. Rydeheard, editors. *Category Theory and Computer Science*, volume 283 of *Lect. Notes in Comp. Sc.* Springer Verlag, 1987.

[28] J.C. Reynolds. Using category theory to design implicit conversions and generic operators. In N.D. Jones, editor, *Proceedings of the Aarhus Workshop on Semantics-Directed Compiler Generation*, volume 94 of *Lect. Notes in Comp. Sc.* Springer Verlag, 1980.

[29] D.E. Rydeheard and R.M. Burstall. *Computational Category Theory*. Prentice Hall, 1988.

[30] M. Spivey. A categorical approach to the theory of lists. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*, number 375 in Lect. Notes in Comp. Sc. Springer Verlag, 1989.