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 calculational properties brought forward by initiality.

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

1 Introduction

Category Theory [ML71] 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, for example [Hoa89, BW90, PAPR86, PPR87, GS89, RB88]. 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 [Hag87b, Hag87a], Reynolds [Rey80] and Cousineau et al [CCM85]) and program derivation (e.g., Malcolm [Mal90a, Mal90b], Meijer [Mei92], Paterson [Pat90], and Fokkinga [Fok92]).

In this chapter 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.

[*] Published in: Formal Aspects of Computing, vol. 4, number 2, pages 673–692, 1992
[†] Research done while at the CWI, Amsterdam.
Current address: University of Twente , dept. INF, P.O. Box 217, NL 7500 AE Enschede, The Netherlands; e-mail: fokkinga@cs.utwente.nl

1 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, has to be modeled as a system of arrows. This may pose serious problems to the newcomer; Arbib and Manes [AM75] 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. 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 mathematical constructions can be characterised.

2 Diagrams and diagram chasing. The basic task in a categorical proof is to show the existence of an arrow, or to show the equality of two arrows, when some other arrows and objects are given. There are several reasons why diagrams may be helpful, 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.

1. *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.

2. *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 for use 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 the initiality of *such-and-so* ."

3. *Commuting diagrams.* Equality of arrows can be indicated pictorially if, by convention, in the picture each two (composite) arrows with the same source and the same target are equal. 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.

4. *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, or constructing an arrow from others. 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.

The calculational style presented in this paper is an alternative to diagram chasing (4). The use of a standard notation for various unique arrows obviates in some cases the need for pictures for the purpose of naming (2). The need for a pictorial overview of the typing (1) is decreased to some extend by a consistent notation, in particular $f \mathbin{;} g$ for composition (so that $f\colon\ a \to b \wedge g\colon\ b \to c \Rightarrow f \,;\, g\colon\ a \to c$ falls out naturally), and some specific notations for arrows that reflect their calculational properties. As regards the presentation of several equalities simultaneously by one commuting diagram (3) we remark that often there is just one equation of interest, the theorem, so that all others are merely auxiliary for the proof or construction only, and there is no reason to display them all at once.

3 The format of a calculation. We present a calculation in the way we have actually derived it (or would like to have derived it). The task of a calculation is to find a definition for some, possibly none, unknowns and to prove an equation or equivalence that contains the unknowns. In general we start with the main task and *reduce* it step by step to simpler tasks, until we finally arrive at *true*. In each step we apply a known *fact*, or *define* an unknown possibly in terms of new unknows, or, in order to proceed, *assume* that some property holds. In the end, all the definitions made along the way constitute a construction of the unknowns, and the assumptions remain as premises that imply the validity of the start equation or equivalence.

Sometimes a calculation can more elegantly be conducted and presented as a transformation between the left hand side of the equality (or equivalence) and the right hand side, using equalities (or equivalences) only. In such a case we usually start with the more complicated side, 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 [DS90] discuss this style in detail, and attribute the calculational format to Feijen, and van Gasteren [Gas88].

4 Calculation with initiality. There is no problem in presenting the pasting of two diagrams as a calculation. For example, pasting 'squares'

$$\text{(a)}\quad \varphi \,;\, f \;=\; p \,;\, \psi \qquad \text{and} \qquad \text{(b)}\quad \psi \,;\, g \;=\; q \,;\, \chi$$

along 'edge' ψ yields $\varphi \,;\, f \,;\, g = p \,;\, q \,;\, \chi$. This is rendered in a one or two step calculation as follows.

$$\begin{array}{ll}
 & \varphi \,;\, f \,;\, g = p \,;\, q \,;\, \chi \\
\equiv & \qquad \text{in left hand side (a): } \varphi \,;\, f = p \,;\, \psi, \\
 & \qquad \text{in right hand side (b)}^{\cup}\text{: } q \,;\, \chi = \psi \,;\, g \\
 & p \,;\, \psi \,;\, g = p \,;\, \psi \,;\, g \\
\equiv & \qquad \text{reflexivity of equality} \\
 & \mathit{true}.
\end{array}$$

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, for all b,

5 $$\exists(x :: \ x\colon\ a \to b \quad \wedge \quad \forall(y :: \ y\colon\ a \to b \ \Rightarrow \ y = x)).$$

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, for all b and x,

$$x\colon\ a \to b \quad \equiv \quad x = \mathcal{F}b\,.$$

Indeed, substituting $x = \mathcal{F}b$ gives $\mathcal{F}b\colon\ a \to b$ (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. The use of equivalences to characterise initiality (and more generally, universality) has been thoroughly advocated by Hoare [Hoa89]. As far as we know, Malcolm [Mal90a] was the first to use this style of reasoning in a formal way for the derivation of functional programs over initial algebras.

6 Functional programming. In this text all arrows (in the sequel called morphisms) in the "base" category 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 specifications that are a bit unusual. The combinations and transformations of morphisms (functions) are fully in the spirit of Backus [Bac78] and Meertens [Mee86]. One should note that nowhere in this text 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, for example by Backus [Bac78] and Hughes [Hug90].

7 Overview. The remainder of the chapter is organised as follows. We assume that the reader is familiar with the very basics of category theory; Goldblatt [Gol79, Chapters 2,3,9], Barr [BW90], and Pierce [Pie91] give a good introduction. In the next section we discuss initiality, and give some preliminaries. Then we specialise the laws for initiality to products and sums in Section 3, to initial algebras in Section 4, to coequalisers and kernel pairs in Section 5, and to colimits in general in Section 6. Each of these sections 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 algebraic 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

8 Nomenclature. Variables $\mathcal{A}, \mathcal{B}, \mathcal{C}$ denote categories, and capital letters vary over functors. Formula $f\colon\ a \to_{\mathcal{A}} b$ means that f is a morphism in $\mathcal{A}$ with source a and target b ($\mathrm{src}_{\mathcal{A}}\, f = a$ and $\mathrm{tgt}_{\mathcal{A}}\, f = b$). In each section, the category that is the "universe of discourse" is denoted $\mathcal{C}$; it may underly the construction of several others. Unless stated otherwise, a morphism is a morphism in $\mathcal{C}$, and similarly for objects. We shall mostly suppress mentioning of the category $\mathcal{C}$, certainly where it would occur as a subscript. In particular, we almost always write $\to$ for $\to_{\mathcal{C}}$. By default, $a, b, c, \ldots$ vary over objects in $\mathcal{C}$, $f, g, h, j, \ldots, p, q, \ldots, x, y, \ldots$ over morphisms in $\mathcal{C}$, and $\gamma, \delta, \ldots$ over natural transformations in $\mathcal{C}$. We denote composition arrows of category $\mathcal{C}$ (and all the categories that inherit its composition) in diagrammatic order: if $f\colon a \to b$ and $g\colon b \to c$ then $f \mathbin{;} g\colon a \to c$. Composition of functors and other mappings is denoted by juxtaposition: $(FG)f = F(Gf)$. We assume in each formula that the free variables are typed in such a way that the formula makes sense, that is, the targets and sources match at each composition and objects and morphisms are in the appropriate category.

9 Categories built upon $\mathcal{C}$. Often an interesting construction in $\mathcal{C}$ can be characterised by initiality in a category $\mathcal{A}$ built upon $\mathcal{C}$. We say $\mathcal{A}$ is **built upon** $\mathcal{C}$ if: each morphism of $\mathcal{A}$ is a -special- morphism in $\mathcal{C}$ and $\mathcal{A}$'s composition and identities are that of $\mathcal{C}$. So, $\mathcal{A}$ is fully determined by defining its objects and morphisms. Moreover, 'built upon' is a reflexive and transitive relation. Here are some examples that we'll meet in the sequel; skip the description upon first reading. (The categorician may recognise $\bigvee(D)$ as the category of *cocones* for the *diagram* D. Dually, the category of *cones* for D is denoted $\bigwedge(D)$. I owe these notations, and those for D below, to Jaap van der Woude.)

Category $\bigvee(\vec{a})$, where $\vec{a}$ is an n-tuple of objects in $\mathcal{C}$. An object in $\bigvee(\vec{a})$ is: an n-tuple of morphisms in $\mathcal{C}$ with a common target and the objects $\vec{a}$ as sources, as suggested by the symbol $\bigvee$ for the 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 $\mathcal{C}$ satisfying $f_i \mathbin{;} x = g_i$ for each index i of the n-tuple. It follows that $x\colon \mathrm{tgt}\, \vec{f} \to \mathrm{tgt}\, \vec{g}$. (As a special case, category $\bigvee(a)$ is known as the co-slice category 'under a', usually denoted $a/\mathcal{C}$.) An object in $\bigvee(a, a)$ is a *parallel pair* with source a.

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

Category $\bigvee(f \ulcorner g)$, where f, g are morphisms in $\mathcal{C}$ with a common source. An object in $\bigvee(f \ulcorner g)$ is: a tuple (h, j) satisfying $f \mathbin{;} h = g \mathbin{;} j$. Let (h, j) and (k, l) be two objects; then a morphism from (h, j) to (k, l) in $\bigvee(f \ulcorner g)$ is: a morphism x in $\mathcal{C}$ satisfying $h \mathbin{;} x = k$ and $j \mathbin{;} x = l$. (This category is used to define the pushout of f, g.)

Category $\mathcal{A}\mathit{lg}(F)$, where F is an endofunctor on $\mathcal{C}$. An object φ of $\mathcal{A}\mathit{lg}(F)$ is: a morphism $\varphi\colon Fa \to a$ in $\mathcal{C}$, for some a called the *carrier* of φ. Let φ and ψ be such objects; then a morphism from φ to ψ in $\mathcal{A}\mathit{lg}(F)$ is: a morphism x in $\mathcal{C}$ satisfying $\varphi \mathbin{;} x = Fx \mathbin{;} \psi$. It follows that $x\colon \mathrm{carrier}\, \varphi \to \mathrm{carrier}\, \psi$, and $\mathrm{carrier}\, \varphi = \mathrm{tgt}\, \varphi$. An object φ is called an F-*algebra*, and a morphism x is called an F-*homomorphism*.

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

$$x\colon\ a \to_{\mathcal{A}} b \qquad \equiv \qquad x = \llparenthesis a \to b \rrparenthesis_{\mathcal{A}} \tag{Charn}$$

Here $\llparenthesis a \to b \rrparenthesis_{\mathcal{A}}$ is just a notation, a name, for a morphism (depending on $\mathcal{A}, a$ 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 ($\mathcal{A}$ 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 $x := \llparenthesis a \to b \rrparenthesis_{\mathcal{A}}$, it follows that for each b there is a morphism from a to b. Thus $\llparenthesis\ \rrparenthesis$ 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 $\mathcal{A}$ is clear from the context we write $\llparenthesis a \to b \rrparenthesis$ rather than $\llparenthesis a \to b \rrparenthesis_{\mathcal{A}}$. It often happens that one initial object in $\mathcal{A}$ is fixed, and in that case $\llparenthesis b \rrparenthesis$ abbreviates $\llparenthesis a \to b \rrparenthesis$. The usual notation for $\llparenthesis b \rrparenthesis_{\mathcal{A}}$ is $!_b$ or $¡_b$. The $!$-notation doesn't work well for categories built upon $\mathcal{A}$ since the notation of a and b may become too large for a subscript.

Finality is dual to initiality; an object a is **final** if: for each object b there exists precisely one morphism from b to a. The default notation for this unique morphism is $\llbracket b \to a \rrbracket_{\mathcal{A}}$, and the characterisation reads

$$x\colon\ b \to_{\mathcal{A}} a \qquad \equiv \qquad x = \llbracket b \to a \rrbracket_{\mathcal{A}}\,.$$

11 Corollaries. Here are some consequences of Charn. A substitution for x such that the right-hand side becomes true yields Self, and a substitution for b, x such that the left-hand side becomes true yields Id:

$$\llparenthesis a \to b \rrparenthesis_{\mathcal{A}}\colon\ a \to_{\mathcal{A}} b \tag{Self}$$

$$id_a = \llparenthesis a \to a \rrparenthesis_{\mathcal{A}} \tag{Id}$$

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

$$x, y\colon\ a \to_{\mathcal{A}} b \qquad \Rightarrow \qquad x = y \tag{Uniq}$$

$$x\colon\ b \to_{\mathcal{A}} c \qquad \Rightarrow \qquad \llparenthesis a \to b \rrparenthesis_{\mathcal{A}}\, ;\, x = \llparenthesis a \to c \rrparenthesis_{\mathcal{A}} \tag{Fusion}$$

The 'proof' of Uniq is left to the reader. For Fusion we argue (suppressing $\mathcal{A}$ and a):

$$\begin{aligned}
&\quad \llparenthesis b \rrparenthesis \,;\, x = \llparenthesis c \rrparenthesis \\
\equiv &\qquad \text{Charn}[\,b, x := c, \llparenthesis b \rrparenthesis \,;\, x\,] \\
&\quad \llparenthesis b \rrparenthesis \,;\, x\colon\ a \to c \\
\Leftarrow &\qquad \text{composition} \\
&\quad \llparenthesis b \rrparenthesis\colon\ a \to b \quad \wedge \quad x\colon\ b \to c \\
\equiv &\qquad \text{Self, and premise}
\end{aligned}$$

$true$.

These five laws become much more interesting in case category $\mathcal{A}$ is built upon another one, and $\to_{\mathcal{A}}$ 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 $x\colon\ b \to_{\mathcal{A}} c$ boils down to the equation $c = b \mathbin{;} x$ (which is the case when $\mathcal{A} = \bigvee(a)$), law Fusion can be formulated as an unconditional equation (by substituting $c := b \mathbin{;} x$ in the consequent, giving $(\!|b|\!) \mathbin{;} x = (\!|b \mathbin{;} x|\!)$). 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.

12 Well-formedness condition. In general, when $\mathcal{A}$ is built upon another category, $\mathcal{C}$ say, the well-formedness condition for the notation $(\!|b|\!)$ is that b (viewed as a composite entity in the underlying category $\mathcal{C}$) is an object in $\mathcal{A}$; this is not a purely syntactic condition.

$$b \text{ in } \mathcal{A} \quad\Rightarrow\quad (\!|a \to b|\!)_{\mathcal{A}} \text{ is a morphism in } \mathcal{C} \qquad\qquad \textsc{Type}$$

In the sequel we adhere to the (dangerous?) convention that in each law the free variables are quantified in such a way that the well-formedness condition, the premise of Type, is met.

13 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 $(\!|a \to b|\!)$ and $(\!|b \to a|\!)$ establish the isomorphism and are unique in doing so. By Type and Self they have the correct typing. We shall show

$$x = (\!|a \to b|\!) \ \wedge\ y = (\!|b \to a|\!) \qquad\equiv\qquad x \mathbin{;} y = id_a \ \wedge\ y \mathbin{;} x = id_b\,,$$

that is, both compositions of $(\!|a \to b|\!)$ and $(\!|b \to a|\!)$ are the identity, and conversely the identities can be factored only in this way. We prove both implications of the equivalence at once.

$$\begin{array}{ll}
 & x = (\!|a \to b|\!) \qquad \wedge \qquad y = (\!|b \to a|\!) \\
\equiv & \qquad \textsc{Charn} \\
 & x\colon\ a \to b \qquad \wedge \qquad y\colon\ b \to a \\
\equiv & \qquad \text{composition} \\
 & x \mathbin{;} y\colon\ a \to a \qquad \wedge \qquad y \mathbin{;} x\colon\ b \to b \\
\equiv & \qquad \textsc{Charn} \\
 & x \mathbin{;} y = (\!|a \to a|\!) \qquad \wedge \qquad y \mathbin{;} x = (\!|b \to b|\!) \\
\equiv & \qquad \textsc{Id} \\
 & x \mathbin{;} y = id_a \qquad \wedge \qquad y \mathbin{;} x = id_b.
\end{array}$$

The equality $(\!|a \to b|\!) ; (\!|b \to a|\!) = id_a$ can be proved 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 Sums

Products and sums are categorical concepts that, specialised to category $\mathcal{S}et$, yield the well-known notions of cartesian product and disjoint union. (In other categories products and sums may get a different interpretation.)

14 Sum. Let $\mathcal{C}$ be arbitrary, and let a, b be objects. A **sum** of a and b is: an initial object in $\bigvee(a, b)$; it may or may not exist. Let inl, inr be a sum of a and b; their common target is denoted $a + b$. We abbreviate $(\!|inl, inr \to f, g|\!)_{\bigvee(a,b)}$ to just $f \triangledown g$, not mentioning the dependency on a, b and inl, inr. (The usual categorical notation for $f \triangledown g$ is $[f, g]$.)

$$f\colon\ a \to c \ \wedge\ g\colon\ b \to c \quad \Rightarrow \quad f \triangledown g\colon\ a + b \to c \qquad \triangledown\text{-TYPE}$$

Working out $\to_{\bigvee(a,b)}$ in terms of equations in $\mathcal{C}$, morphisms inl, inr and operation $\triangledown$ are determined ("up to isomorphism") by law CHARN, and consequently also satisfy the other laws.

$$inl ; x = f \ \wedge\ inr ; x = g \quad \equiv \quad x = f \triangledown g \qquad \triangledown\text{-CHARN}$$

$$inl ; f \triangledown g = f \ \wedge\ inr ; f \triangledown g = g \qquad \triangledown\text{-SELF}$$

$$inl \triangledown inr = id \qquad \triangledown\text{-ID}$$

$$inl ; x = inl ; y \ \wedge\ inr ; x = inr ; y \quad \Rightarrow \quad x = y \quad (\text{“jointly epic”}) \qquad \triangledown\text{-UNIQ}$$

$$f ; x = h \ \wedge\ g ; x = j \quad \Rightarrow \quad f \triangledown g ; x = h \triangledown j \qquad \triangledown\text{-FUSION}$$

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 be done tacitly in the sequel.

15 Products. Products are, by definition, dual to sums. Let exl, exr be a product of a and b, supposing one exists; its common source is denoted $a \times b$. We abbreviate $[\![f, g \to exl, exr]\!]_{\bigwedge(a,b)}$ to just $f \vartriangle g$. (The usual categorical notation for $f \vartriangle g$ is $\langle f, g\rangle$).

$$f\colon\ c \to a \ \wedge\ g\colon\ c \to b \quad \Rightarrow \quad f \vartriangle g\colon\ c \to a \times b \qquad \vartriangle\text{-TYPE}$$

The laws for exl, exr and $\vartriangle$ work out as follows:

$$x ; exl = f \ \wedge\ x ; exr = g \quad \equiv \quad x = f \vartriangle g \qquad \vartriangle\text{-CHARN}$$

$$f \vartriangle g \mathbin{;} \mathit{exl} = f \;\wedge\; f \vartriangle g \mathbin{;} \mathit{exr} = g \qquad \vartriangle\text{-}\textsc{Self}$$

$$\mathit{exl} \vartriangle \mathit{exr} = id \qquad \vartriangle\text{-}\textsc{Id}$$

$$x \mathbin{;} \mathit{exl} = y \mathbin{;} \mathit{exl} \;\wedge\; x \mathbin{;} \mathit{exr} = y \mathbin{;} \mathit{exr} \quad\Rightarrow\quad x = y \quad (\text{“jointly monic”}) \qquad \vartriangle\text{-}\textsc{Uniq}$$

$$x \mathbin{;} f \vartriangle g = (x \mathbin{;} f) \vartriangle (x \mathbin{;} g) \qquad \vartriangle\text{-}\textsc{Fusion}$$

16 Application. As a first application we show that *inl* is monic. (Morphism f is monic if: $x \mathbin{;} f = y \mathbin{;} f \Rightarrow x = y$.) By symmetry it follows that *inr* is monic too, and dually each of *exl* and *exr* is epic.

$$\begin{array}{ll}
 & x = y \\
\equiv & \quad \text{aiming at “ } \mathbin{;} \mathit{inl} \text{” after } x \text{ and } y \text{, use } \triangledown\text{-}\textsc{Self}[\, f := id\,] \\
 & x \mathbin{;} \mathit{inl} \mathbin{;} id \triangledown g = y \mathbin{;} \mathit{inl} \mathbin{;} id \triangledown g \\
\Leftarrow & \quad \text{Leibniz} \\
 & x \mathbin{;} \mathit{inl} = y \mathbin{;} \mathit{inl}
\end{array}$$

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

As a second application we show that $\triangledown$ and $\vartriangle$ abide. Two binary operations $\ominus$ and $\oplus$ **abide** with each other if: for all values a, b, c, d

$$(a \ominus b) \oplus (c \ominus d) \quad = \quad (a \oplus c) \ominus (b \oplus d)\,.$$

Writing $a \ominus b$ as $\frac{a}{b}$ and $a \oplus b$ as $a \,|\, b$, the equation reads

$$\frac{a}{b} \,\Big|\, \frac{c}{d} \quad = \quad \frac{a \,|\, c}{b \,|\, d}\,.$$

The term *abide* has been coined by Bird [Bir89] and comes from “above-beside.” In category theory this property is called the ‘middle exchange rule.’

$$\begin{array}{ll}
 & (f \triangledown g) \vartriangle (h \triangledown j) = (f \vartriangle h) \triangledown (g \vartriangle j) \\
\equiv & \quad \triangledown\text{-}\textsc{Charn}\,[x, f, g := \text{lhs},\; f \vartriangle h,\; g \vartriangle j] \\
 & \mathit{inl} \mathbin{;} (f \triangledown g) \vartriangle (h \triangledown j) = f \vartriangle h \quad \wedge \quad \mathit{inr} \mathbin{;} (f \triangledown g) \vartriangle (h \triangledown j) = g \vartriangle j \\
\equiv & \quad \vartriangle\text{-}\textsc{Fusion} \text{ (at two places)} \\
 & (\mathit{inl} \mathbin{;} f \triangledown g) \vartriangle (\mathit{inl} \mathbin{;} h \triangledown j) = f \vartriangle h \quad \wedge \quad (\mathit{inr} \mathbin{;} f \triangledown g) \vartriangle (\mathit{inr} \mathbin{;} h \triangledown j) = g \vartriangle j \\
\equiv & \quad \triangledown\text{-}\textsc{Self} \text{ (at four places)} \\
 & f \vartriangle h = f \vartriangle h \quad \wedge \quad g \vartriangle j = g \vartriangle j \\
\equiv & \quad \text{equality} \\
 & \mathit{true}\,.
\end{array}$$

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

$$f + g \quad = \quad (f \mathbin{;} \mathit{inl}) \triangledown (g \mathbin{;} \mathit{inr}) \quad : \quad a + c \to b + d$$

$$f \times g \quad = \quad (\mathit{exl} \mathbin{;} f) \vartriangle (\mathit{exr} \mathbin{;} g) \quad : \quad a \times c \to b \times d\,.$$

These $+$ and $\times$ are bifunctors: $\mathit{id} + \mathit{id} = \mathit{id}$ and $f + g \mathbin{;} h + j = (f \mathbin{;} h) + (g \mathbin{;} j)$, and similarly for $\times$.

4 Algebras

Let F be a functor from $\mathcal{C}$ to $\mathcal{C}$. Recall from Section 2 that an F-algebra is just a morphism $\varphi\colon Fa \to a$ for some a, called the carrier of φ, and a homomorphism from φ to ψ is a morphism f satisfying $\varphi \mathbin{;} f = Ff \mathbin{;} \psi$. It follows that $f\colon \text{carrier}\,\varphi \to \text{carrier}\,\psi$. We abbreviate $\to_{\mathcal{A}\mathit{lg}(F)}$ to just $\to_F$. To motivate the terminology, consider the following two cases.

A single binary operation $\varphi\colon a \times a \to a$ is an F-algebra, where functor F is defined by $Fx = x \times x$. Then '$f\colon \varphi \to_F \psi$' means $\varphi \mathbin{;} f = f \times f \mathbin{;} \psi$, which is a slight generalisation of the property that f distributes over φ. Now consider two algebras (or n-ary operations) of different type but with the same carrier, say $\varphi_i\colon F_i a \to a$ for $i = 0, 1$. We can then form $\varphi = \varphi_0 \triangledown \varphi_1\colon F_0 a + F_1 a \to a$; 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 $\varphi_0 = \mathit{inl} \mathbin{;} \varphi$ and $\varphi_1 = \mathit{inr} \mathbin{;} \varphi$ (see law $\triangledown$-Charn). Statement $f\colon \varphi_0 \triangledown \varphi_1 \to_{F_0 + F_1} \psi_0 \triangledown \psi_1$ boils down to the two statements $f\colon \varphi_i \to_{F_i} \psi_i$ for $i = 0, 1$. Thus, classical single-sorted algebras can be modeled as F-algebras. Fokkinga [Fok91, Fok92] shows how to deal with equations (like associativity), many-sortedness, and more general datatypes in this frame-work.

Let α be initial in $\mathcal{A}\mathit{lg}(F)$ (supposing it exists). We fix this α throughout what follows, and we write $(\!|\varphi|\!)$ for $(\!|\alpha \to \varphi|\!)_{\mathcal{A}\mathit{lg}(F)}$, and call it a *catamorphism*. This notation supposes that φ is an F-algebra:

$$\varphi \text{ is an } F\text{-algebra} \quad \Rightarrow \quad (\!|\varphi|\!)\colon \text{carrier}\,\alpha \to \text{carrier}\,\varphi\,. \qquad \text{cata-Type}$$

Then the laws for α and $(\!|\ |\!)$ work out as follows.

$$\alpha \mathbin{;} x = Fx \mathbin{;} \varphi \quad \equiv \quad x = (\!|\varphi|\!) \qquad \text{cata-Charn}$$

$$\alpha \mathbin{;} (\!|\varphi|\!) = F\,(\!|\varphi|\!) \mathbin{;} \varphi \qquad \text{cata-Self}$$

$$\mathit{id} = (\!|\alpha|\!) \qquad \text{cata-Id}$$

$$\alpha \mathbin{;} x = Fx \mathbin{;} \varphi \ \wedge\ \alpha \mathbin{;} y = Fy \mathbin{;} \varphi \quad \Rightarrow \quad x = y \qquad \text{cata-Uniq}$$

$$\varphi \mathbin{;} x = Fx \mathbin{;} \psi \quad \Rightarrow \quad (\!|\varphi|\!) \mathbin{;} x = (\!|\psi|\!) \qquad \text{cata-Fusion}$$

Let a be the carrier of α. Below we shall show that α is an isomorphism: $\alpha\colon Fa \to a$, the inverse of which we denote $\alpha^{\cup}$. We might call α a "constructor", since in $\mathcal{S}\mathit{et}$ 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 $x = \alpha^{\cup} \mathbin{;} Fx \mathbin{;} \varphi$. Thus cata-CHARN says that the "inductive definition" $x = \alpha^{\cup} \mathbin{;} Fx \mathbin{;} \varphi$ 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 $x = \alpha^{\cup} \mathbin{;} Fx \mathbin{;} \varphi$ and $y = \alpha^{\cup} \mathbin{;} Fy \mathbin{;} \varphi$, 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 $(\!|\varphi|\!) \mathbin{;} x$ can be expressed as a single catamorphism.

17 Application. As an application we show that an initial F-algebra α is —up to isomorphism— a fixed point of F, that is, $F\alpha \cong \alpha$ in $\mathcal{A}lg(F)$. To prove this, we have to establish a pair x, y of morphisms in $\mathcal{A}lg(F)$ (F-algebra homomorphisms in $\mathcal{C}$),

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

that are each other's inverse. Law cata-CHARN implies that y is $(\!|F\alpha|\!)$. We start proving that $y = (\!|F\alpha|\!)$ is a "pre-inverse" of x, deriving along the way a definition for x. For this we argue:

$$\begin{array}{ll} & y \mathbin{;} x = id_\alpha \\ \equiv & \qquad \text{definition } y \\ & (\!|F\alpha|\!) \mathbin{;} x = id \\ \equiv & \qquad \text{cata-ID} \\ & (\!|F\alpha|\!) \mathbin{;} x = (\!|\alpha|\!) \\ \Leftarrow & \qquad \text{cata-FUSION} \\ & F\alpha \mathbin{;} x = Fx \mathbin{;} \alpha \\ \equiv & \qquad \textbf{define } x = \alpha \\ & \mathit{true}. \end{array}$$

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

$$\begin{array}{ll} & x \mathbin{;} y \\ = & \qquad \text{definition } x, y \\ & \alpha \mathbin{;} (\!|F\alpha|\!) \\ = & \qquad \text{cata-SELF} \\ & F(\!|F\alpha|\!) \mathbin{;} F\alpha \\ = & \qquad \text{functor, above: } (\!|F\alpha|\!) \text{ is pre-inverse of } \alpha \\ & F\mathit{id} \\ = & \qquad \text{functor} \\ & \mathit{id}. \end{array}$$

Since $\mathcal{A}lg(F)$ is built upon $\mathcal{C}$ a corollary of the isomorphism $\alpha\colon F\alpha \cong \alpha$ in $\mathcal{A}lg(F)$ is the isomorphism $\alpha\colon Fa \cong a$ in $\mathcal{C}$, where $a = \text{carrier}\,\alpha$.

5 Coequalisers and Kernel pairs

Coequalisers and kernel pairs are categorical notions that, specialised to category $\mathcal{S}et$, yield the well-known notion of induced equivalence relation, and the lesser known notion of induced kernel pair, respectively. We shall present the properties of coequalisers and kernel pairs in a way suitable for calculation.

18 Coequalisers. Let $\mathcal{C}$ be arbitrary, and let (f, g) be a parallel pair. A **coequaliser** of (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 $(\!|p \to q|\!)_{\bigvee(f\|g)}$ since, as we shall explain, the fraction notation better suggests the calculational properties.

$$f \,;\, q = g \,;\, q \quad\Rightarrow\quad p\backslash q\colon\ \text{tgt}\,p \to \text{tgt}\,q \qquad \backslash\text{-Type}$$

Then the laws for p and $\backslash$ work out as follows.

$$\begin{array}{llcll}
 & p \,;\, x = q & \equiv & x = p\backslash q & \backslash\text{-Charn}\\
 & p \,;\, p\backslash q = q & & & \backslash\text{-Self}\\
 & id = p\backslash p & & & \backslash\text{-Id}\\
 & p \,;\, x = q \ \wedge\ p \,;\, y = q & \Rightarrow & x = y & \backslash\text{-Uniq}\\
\text{i.e.,} & p \,;\, x = p \,;\, y & \Rightarrow & x = y & (p \text{ epic})\\
 & q \,;\, x = r & \Rightarrow & p\backslash q \,;\, x = p\backslash r & \backslash\text{-Fusion}\\
\text{i.e.,} & p\backslash q \,;\, x = p\backslash(q \,;\, x) & & &
\end{array}$$

In accordance with the convention explained in paragraph 12 we have omitted in laws \-Charn, \-Self and \-Fusion the well-formedness condition that q is an object in $\bigvee(f\|g)$; the notation $...\backslash q$ is only senseful if $f \,;\, q = g \,;\, q$, like in arithmetic where the notation m/n is only senseful if n differs from 0. Notice also how \-Fusion simplifies to an unconditional fusion law. Similarly law \-Uniq simplifies to the assertion that each 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 \ 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.

19 Additional laws. The following law confirms the choice of notation once more.

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

Here is one way to prove it.

$p\backslash q \mathbin{;} q\backslash r$

$=$ $\backslash$-FUSION

$p\backslash(q \mathbin{;} q\backslash r)$

$=$ $\backslash$-SELF

$p\backslash r$.

An interesting aspect is that the omitted 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 $\backslash$-COMPOSE reads:

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

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

$(\!|a \to b|\!)_{\mathcal{A}} \mathbin{;} (\!|b \to c|\!)_{\mathcal{B}} = (\!|a \to c|\!)_{\mathcal{A}}$

$\Leftarrow$ FUSION

$(\!|b \to c|\!)_{\mathcal{B}}\colon\ b \to_{\mathcal{A}} c$

$\equiv$ both $\mathcal{A}$ and $\mathcal{B}$ are full subcategories of $\mathcal{C}$,
each containing both b and c

$(\!|b \to c|\!)_{\mathcal{B}}\colon\ b \to_{\mathcal{B}} c$

$\equiv$ SELF

true.

Another law that we shall use below has to do with functors. As before, let p be a coequaliser. Then

$$F(p\backslash q) = Fp\backslash Fq \qquad \backslash\text{-FCTR}$$

The implicit well-formedness condition here is that Fp is a coequaliser. Clearly, this condition is satisfied when F preserves coequalisers. The proof of the law reads:

$F(p\backslash q) = Fp\backslash Fq$

$\equiv$ $\backslash$-CHARN

$Fp \mathbin{;} F(p\backslash q) = Fq$

$\equiv$ functor

$F(p \mathbin{;} p\backslash q) = Fq$

$\equiv$ $\backslash$-SELF

true.

20 Kernel pairs. Let $\mathcal{C}$ be arbitrary, and let p be a morphism. A **kernel pair** of p is: a final object in $\bigwedge(p \lrcorner p)$. (Category $\bigvee(f \lrcorner g)$ is the dual of $\bigwedge(f \ulcorner g)$ explained in Section 2.) Let (f, g) be a kernel pair of p, supposing one exists. This time we use the notation $(d,e)/_p(f,g)$ or simply $(d,e)/(f,g)$ instead of $[\![d, e \to f, g]\!]_{\bigwedge(p \lrcorner p)}$.

$$d \mathbin{;} p = e \mathbin{;} p \quad \Rightarrow \quad (d,e)/(f,g)\colon\ \operatorname{src} d \to \operatorname{src} f \;=\; \operatorname{src} e \to \operatorname{src} g \qquad /\text{-}\textsc{Type}$$

Then the laws for (f,g) and $/$ work out as follows.

$$d = x \mathbin{;} f \ \wedge\ e = x \mathbin{;} g \qquad \equiv \qquad x = (d,e)/(f,g) \qquad /\text{-}\textsc{Charn}$$

$$d = (d,e)/(f,g) \mathbin{;} f \quad \wedge \quad e = (d,e)/(f,g) \mathbin{;} g \qquad /\text{-}\textsc{Self}$$

$$id = (f,g)/(f,g) \qquad /\text{-}\textsc{Id}$$

$$\left.\begin{array}{l} d = x \mathbin{;} f \ \ \wedge\ \ e = x \mathbin{;} g \\ d = y \mathbin{;} f \ \ \wedge\ \ e = y \mathbin{;} g \end{array}\right\} \qquad \Rightarrow \qquad x = y \qquad /\text{-}\textsc{Uniq}$$

i.e., $$x \mathbin{;} f = y \mathbin{;} f \quad \wedge \quad x \mathbin{;} g = y \mathbin{;} g \qquad \Rightarrow \qquad x = y \qquad \text{(jointly monic)}$$

$$x \mathbin{;} (d,e)/(f,g) = (x \mathbin{;} d,\ x \mathbin{;} e)/(f,g) \qquad /\text{-}\textsc{Fusion}$$

$$(d,e)/(f,g) \mathbin{;} (f,g)/(h,j) = (d,e)/(h,j) \qquad /\text{-}\textsc{Compose}$$

$$F((d,e)/(f,g)) = F(d,e)/F(f,g) \qquad /\text{-}\textsc{Fctr}$$

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

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

$$\begin{array}{lcll} C(f,g) & = & \text{'the' coequaliser of } f, g & \text{for } (f,g) \text{ in } \bigwedge(a,a) \\ Kp & = & \text{'the' kernel pair of } p & \text{for } p \text{ in } \bigvee(a)\,. \end{array}$$

We shall extend them to functors $C\colon \bigwedge(a,a) \to \bigvee(a)$ and $K\colon \bigvee(a) \to \bigwedge(a,a)$, and then prove that they form an adjunction.

To define Cx for a morphism x in $\bigwedge(a,a)$ we make an obvious choice.

(a) $$Cx \quad = \quad C(d,e)\backslash C(f,g) \qquad \text{for } x\colon\ (d,e) \to (f,g) \text{ in } \bigwedge(a,a)\,.$$

It remains to prove that C is a functor. Since in general $p\backslash q\colon\ p \to q$ (in the appropriate category, see $\backslash$-Type), it is immediate that Cx above has the right type, namely $C(d,e) \to C(f,g)$ in $\bigvee(a,a)$. The two functor axioms $C id = id$ and $C(x \mathbin{;} y) = Cx \mathbin{;} Cy$ follow immediately by $\backslash$-Id and $\backslash$-Compose.

To define Ku for a morphism u in $\bigvee(a)$ we make an obvious choice too.

(b) $$Ku \quad = \quad Kp/Kq \qquad \text{for } u\colon\ p \to q \text{ in } \bigvee(a)\,.$$

Thus extended, K is a functor by a similar argument as above.

To prove that C is adjoint to K we establish natural transformations $\varepsilon\colon CK \dot{\to} I$ and $\eta\colon I \dot{\to} KC$ such that $\eta K \mathbin{;} K\varepsilon = \mathit{id}K$ and $C\eta \mathbin{;} \varepsilon C = \mathit{id}C$. Take

$$\varepsilon q \quad = \quad CKq\backslash q \quad : \quad CKq \to_{\bigvee(a)} q \qquad\qquad \text{for all } q \text{ in } \bigvee(a)\ .$$

The naturality of ε is shown as follows. For arbitrary $u\colon p \to_{\bigvee(a)} q$,

$$\begin{array}{ll}
 & CKu \mathbin{;} \varepsilon q \\
= & \qquad \text{definition } C,\ K \text{ and } \varepsilon \text{, noting that } u\colon p \to_{\bigvee(a)} q \\
 & CKp\backslash CKq \mathbin{;} CKq\backslash q \\
= & \qquad \backslash\text{-}\textsc{Compose} \\
 & CKp\backslash q \\
= & \qquad \text{equation ``} u\colon p \to_{\bigvee(a)} q \text{''} \\
 & CKp\backslash(p \mathbin{;} u) \\
= & \qquad \backslash\text{-}\textsc{Fusion} \\
 & CKp\backslash p \mathbin{;} u \\
= & \qquad \text{definition } \varepsilon \text{ and } I \\
 & \varepsilon p \mathbin{;} Iu
\end{array}$$

as desired. Further we take

$$\eta(d,e) \quad = \quad (d,e)/KC(d,e) \quad : \quad (d,e) \to_{\bigwedge(a,a)} KC(d,e)\,.$$

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 $\eta K \mathbin{;} K\varepsilon = \mathit{id}K$. Let q be arbitrary, then

$$\begin{array}{ll}
 & (\eta K \mathbin{;} K\varepsilon)q \\
= & \qquad \text{composition of natural transformations, and definitions of } \eta,\ \varepsilon \\
 & Kq/KCKq \mathbin{;} K(CKq\backslash q) \\
= & \qquad \text{definition } K \text{ (see (b) above with } u,p,q := q\backslash CKq, q, CKq\text{,} \\
 & \qquad \text{noting that } u\colon p \to q \text{ follows from } \backslash\text{-}\textsc{Self}) \\
 & K(q\backslash CKq) \mathbin{;} K(CKq\backslash q) \\
= & \qquad \text{functor, } \backslash\text{-}\textsc{Compose} \\
 & K(q\backslash q) \\
= & \qquad \backslash\text{-}\textsc{Id}\text{, noting that } \mathit{id}_q \text{ in } \bigvee(a) \text{ is } \mathit{id}_{\mathrm{tgt}q} \text{ in } \mathcal{C} \\
 & K(\mathit{id}_q) \\
= & \qquad \text{functor} \\
 & \mathit{id}Kq.
\end{array}$$

The proof of $C\eta \mathbin{;} \varepsilon C = \mathit{id}C$ is again quite similar to the above one.

6 Colimits

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

22 Colimit. In order to avoid a lot of explicit quantifications and subscriptions, which hinder effective calculation, we take a formalisation of colimits by means of natural transformations. Several manipulations on the subscripts can then be phrased as well-known manipulations with natural transformations as a whole.

A **diagram** in $\mathcal{C}$ is a functor $D\colon \mathcal{D} \to \mathcal{C}$, for some category $\mathcal{D}$, called the **shape** of the diagram. (Usually the image $D\mathcal{D}$ in $\mathcal{C}$ is called the diagram.) Category $\bigvee D$ is built upon $\mathcal{C}$ as follows. An object in $\bigvee D$, called **cocone** for D, is: a natural transformation $\gamma\colon D \mathrel{\dot\to} \underline{c}$ for some object c in $\mathcal{C}$ ($\underline{c}$ is the constant functor mapping each morphism f into id_c). Let γ and δ be cocones for D; then a morphism from γ to δ in $\bigvee D$ is: a morphism x in $\mathcal{C}$ satisfying $\gamma \mathbin{;} x = \delta$ (the composition is a slight adaptation of that in $\mathcal{C}$; see below). A **colimit** for D is: an initial object in $\bigvee D$.

The required "commutativity of the triangles" follows from the naturality: for each $Df\colon Da \to Db$ in the 'diagram' $D\mathcal{D}$ in $\mathcal{C}$

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

For natural transformations in general, hence for cocones in particular, the following definitions are standard. For $\gamma\colon D \mathrel{\dot\to} \underline{c}$ and $\delta\colon D \mathrel{\dot\to} \underline{d}$:

- for each $x\colon c \to d$,
  $(\gamma \mathbin{;} x)a = \gamma a \mathbin{;} x$, so that $\gamma \mathbin{;} x\colon D \mathrel{\dot\to} \underline{d}$ is a cocone for D again.
- for each functor $F\colon \mathcal{C} \to \mathcal{C}$,
  $(F\gamma)a = F(\gamma a)$, so that $F\gamma\colon FD \mathrel{\dot\to} F\underline{c}$ 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 by definition $(F\gamma)a = F(\gamma a)$ we omit the parentheses.
- for each functor $S\colon \mathcal{D} \to \mathcal{D}$,
  $(\gamma S)a = \gamma(Sa)$, so that $\gamma S\colon DS \mathrel{\dot\to} \underline{c}$ is a cocone for DS (note that $\underline{c}S = \underline{c}$).
  If S transforms the shape, γS is the transformed cocone.
  Since by definition $(\gamma S)a = \gamma(Sa)$ we omit the parentheses.

23 The laws. Let γ be a colimit for D. We write $\gamma\backslash_D\delta$ or simply $\gamma\backslash\delta$, instead of $(\!|\gamma \to \delta|\!)_{\bigvee D}$. Then the laws for γ and $\backslash$ work out as follows.

$$\delta\colon D \mathrel{\dot\to} \underline{d} \qquad\Rightarrow\qquad \gamma\backslash\delta\colon \operatorname{tgt}\gamma \to d \qquad\qquad \backslash\text{-TYPE}$$

and

$$
\begin{array}{llll}
\gamma \mathbin{;} x = \delta & \equiv & x = \gamma\backslash\delta & \backslash\text{-C}\textsc{harn} \\
\gamma \mathbin{;} \gamma\backslash\delta = \delta & & & \backslash\text{-S}\textsc{elf} \\
\gamma\backslash\gamma = id & & & \backslash\text{-I}\textsc{d} \\
\gamma \mathbin{;} x = \gamma \mathbin{;} y & \Rightarrow & x = y \quad (\gamma \text{ jointly epic}) & \backslash\text{-U}\textsc{niq} \\
\gamma\backslash\delta \mathbin{;} x = \gamma\backslash(\delta \mathbin{;} x) & & & \backslash\text{-F}\textsc{usion} \\
\gamma\backslash\delta \mathbin{;} \delta\backslash\varepsilon = \gamma\backslash\varepsilon & & & \backslash\text{-C}\textsc{ompose} \\
F(\gamma\backslash\delta) = F\gamma\backslash F\delta & & & \backslash\text{-F}\textsc{ctr}
\end{array}
$$

for each D-cocone δ, ε (δ and $F\gamma$ being a colimit when occurring as the left argument of $\backslash$.) Notice also that, by definition of γS and $\backslash$-Self,

24 $\quad \gamma S \mathbin{;} \gamma\backslash\delta \quad = \quad (\gamma \mathbin{;} \gamma\backslash\delta)S \quad = \quad \delta S\,.$

If γS is a colimit then $\gamma S\backslash\delta S$ is well-formed, and the equality $\gamma\backslash\delta = \gamma S\backslash\delta S$ follows by $\backslash$-Charn from the equation.

25 Application. We present the well-known construction of an initial F-algebra. Our interest is *solely* in the algebraic, calculational style of various subproofs. The construction will require that $\mathcal{C}$ has an initial object and a colimit for each ω-chain, and that functor F preserves colimits of ω-chains.

Given endofunctor F we wish to construct an F-algebra, $\alpha\colon Fa \to a$ say, that is initial in $\mathcal{A}lg(F)$. Forgoing initiality for the time being, we derive a construction of an $\alpha\colon Fa \to a$ as follows. (Read the steps and their explanation below in parallel!)

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

Step (a): this is motivated by the wish that α be initial in $\mathcal{A}lg(F)$, and so α will be an isomorphism; in other words, in view of the required initiality the step is *no* strengthening. *Step* (b): here we merely decide that α, a come from a (co)limit construction; this is true for many categorical constructions. So we aim at $\alpha\colon F\gamma \cong \ldots$, where γ is 'the' colimit (which we assume to exist) for a diagram D yet to be defined. Since $F\gamma$ is a FD-cocone, there has to be another FD-cocone on the dots. To keep things simple, we aim at an FD-cocone constructed from γ, say γS, where S is an endofunctor on src$D$. Since γS is evidently a DS-cocone, and must be an FD-cocone, it follows that $FD = DS$ is

another requirement.
Step (c): the hint '$F\gamma$ is colimit for FD' follows from the assumption that F preserves colimits, and 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 other's 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 $\mathcal{A}lg(F)$' 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 $\mathit{0}, S\colon \omega \to \omega$ are defined by $\mathit{0}(i{\leq}j) = 0{\leq}0$ and $S(i{\leq}j) = (i{+}1){\leq}(j{+}1)$.

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

26 $$FD(i{\leq}j) \;=\; FF^i(\!| F^{j-i}\mathit{o} |\!) \;=\; F^{i+1}(\!| F^{(j+1)-(i+1)}\mathit{o} |\!) \;=\; D((i{+}1){\leq}(j{+}1)) \;=\; DS(i{\leq}j)\,.$$

Thanks to the particular form of ω, natural transformations of the form $\varepsilon\colon D \dot{\to} G$ (some G) can be defined by induction, that is, by defining

$$\begin{array}{lll} \varepsilon\mathit{0} & : & D\mathit{0} \dot{\to} G\mathit{0} \qquad \text{or, equivalently} \qquad \varepsilon 0\colon D0 \to G0 \\ \varepsilon S & : & DS \dot{\to} GS\,. \end{array}$$

We shall use this form of definition in Part 2 and Part 3 below.

Part 2. (Proof of 'γS is colimit for DS' where γ is a colimit for D.) Our task is to construct for arbitrary cocone $\delta\colon DS \dot{\to} \underline{d}$ a morphism $(\!| \gamma S \to \delta |\!)_{\bigvee(DS)}$ such that

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

Our guess is that $\gamma\backslash\varepsilon$ may be chosen for $(\!| \gamma S \to \delta |\!)_{\bigvee(DS)}$ for some suitably chosen $\varepsilon\colon D \dot{\to} \underline{d}$ that depends on δ. This guess is sufficient to start the proof of ($\spadesuit$); we shall derive a definition of ε (more specifically, for $\varepsilon 0$ and εS) along the way.

$$\begin{array}{ll} & x = \gamma\backslash\varepsilon \\ \equiv & \qquad \backslash\text{-}\textsc{Charn} \end{array}$$

$\gamma \mathbin{;} x = \varepsilon$

$\equiv$ observation at the end of Part 1

$(\gamma \mathbin{;} x)\mathit{0} = \varepsilon\mathit{0} \quad \wedge \quad (\gamma \mathbin{;} x)S = \varepsilon S$

$\equiv$ ‘standard definition’ for natural transformations (see paragraph 22)

$\gamma 0 \mathbin{;} x = \varepsilon 0 \quad \wedge \quad \gamma S \mathbin{;} x = \varepsilon S$

$\equiv$ { aiming at the left hand side of (♠) }
define $\varepsilon S = \delta$ (noting that δ: $DS \mathrel{\dot{\to}} \underline{d} = DS \mathrel{\dot{\to}} \underline{d}S$)

$\gamma 0 \mathbin{;} x = \varepsilon 0 \quad \wedge \quad \gamma S \mathbin{;} x = \delta$

$(*)$ $\equiv$ define $\varepsilon 0$ below such that $\gamma S \mathbin{;} x = \delta \ \Rightarrow\ \gamma 0 \mathbin{;} x = \varepsilon 0$ for all x

$\gamma S \mathbin{;} x = \delta.$

In order to define $\varepsilon 0$ satisfying the requirement derived at step $(*)$, we calculate

$\gamma 0 \mathbin{;} x$

$=$ { anticipating next steps, introduce an identity }

$\gamma 0 \mathbin{;} \underline{c}(0{\le}1) \mathbin{;} x$

$=$ naturality γ (“commutativity of the triangle”)

$D(0{\le}1) \mathbin{;} \gamma 1 \mathbin{;} x$

$=$ using $\gamma S \mathbin{;} x = \delta$

$D(0{\le}1) \mathbin{;} \delta 0$

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

Part 3. (Proof of ‘ α is initial in $\mathcal{A}lg(F)$ ’ where $\alpha = F\gamma \backslash \gamma S$.) Put $a = \mathrm{tgt}\alpha = \mathrm{tgt}\gamma$ (as we did in the main steps (a), (b), (c) at the start). Let φ: $Fb \to b$ be arbitrary. We have to construct a morphism $(\!|\alpha \rightarrow \varphi|\!)_F$: $a \to b$ in $\mathcal{C}$ such that

(♣) $\quad F\gamma \backslash \gamma S \mathbin{;} x = Fx \mathbin{;} \varphi \quad \equiv \quad x = (\!|\alpha \rightarrow \varphi|\!)_F$.

Our guess is that the required morphism $(\!|\alpha \rightarrow \varphi|\!)_F$ can be written as $\gamma \backslash \delta$ for some suitably chosen D-cocone δ. This guess is sufficient to start the proof of (♣), deriving a definition for δ (more specifically, for $\delta\mathit{0}$ and δS) along the way.

$F\gamma \backslash \gamma S \mathbin{;} x = Fx \mathbin{;} \varphi$

$\equiv$ $\backslash$-Fusion

$F\gamma \backslash (\gamma S \mathbin{;} x) = Fx \mathbin{;} \varphi$

$\equiv$ $\backslash$-Charn$[\gamma, \delta, x := F\gamma,\ \gamma S \mathbin{;} x,\ Fx \mathbin{;} \varphi]$

$F\gamma \mathbin{;} Fx \mathbin{;} \varphi = \gamma S \mathbin{;} x$

$\equiv$ lhs: functor, rhs: ‘standard definition’ for ntrf (see paragraph 22)

$F(\gamma \mathbin{;} x) \mathbin{;} \varphi = (\gamma \mathbin{;} x)S$

$(*)$ $\equiv$ explained and proved below (defining δ)

$$\begin{array}{ll} & \gamma \mathbin{;} x = \delta \\ \equiv & \qquad \backslash\text{-}\textsc{Charn} \\ & x = \gamma\backslash\delta. \end{array}$$

Arriving at the line above $(*)$ I see no way to make progress except to work bottom-up from the last line. Having the lines above and below $(*)$ available, we define δSn in terms of δn by

$$\delta S \quad = \quad F\delta \mathbin{;} \varphi\,,$$

a definition that is also suggested by type considerations alone. Now part $\Leftarrow$ of equivalence $(*)$ is immediate:

$$\begin{array}{ll} & F(\gamma \mathbin{;} x) \mathbin{;} \varphi = (\gamma \mathbin{;} x)S \\ \Leftarrow & \qquad \text{definition } \delta S\colon\ F\delta \mathbin{;} \varphi = \delta S \\ & \gamma \mathbin{;} x = \delta. \end{array}$$

For part $\Rightarrow$ of equivalence $(*)$ we argue as follows, assuming the line above $(*)$ as a premise, and defining $\delta 0$ along the way.

$$\begin{array}{ll} & \gamma \mathbin{;} x = \delta \\ \equiv & \qquad \text{induction principle} \\ & (\gamma \mathbin{;} x)\mathit{0} = \delta \mathit{0} \quad \wedge \quad \forall(n ::\quad (\gamma \mathbin{;} x)n = \delta n \ \Rightarrow\ (\gamma \mathbin{;} x)Sn = \delta Sn) \\ \equiv & \qquad \text{proved below: the `induction base' in (i), and the `induction step' in (ii)} \\ & \mathit{true}. \end{array}$$

For (i), the induction base, we calculate

$$\begin{array}{ll} & \gamma 0 \mathbin{;} x \\ = & \qquad \textsc{Charn}\text{, using } \gamma 0\colon\ \mathit{0} \rightarrow c \\ & (\!|a|\!)_{\mathcal{C}} \mathbin{;} x \\ = & \qquad \textsc{Fusion}\text{, using } x\colon\ a \rightarrow b \\ & (\!|b|\!)_{\mathcal{C}} \\ = & \qquad \textbf{define } \delta 0 = (\!|b|\!)_{\mathcal{C}} \\ & \mathit{true}. \end{array}$$

And for (ii), the induction step, we calculate for arbitrary n, using the induction hypothesis $(\gamma \mathbin{;} x)n = \delta n$,

$$\begin{array}{ll} & (\gamma \mathbin{;} x)Sn \\ = & \qquad \text{line above } (*) \\ & (F(\gamma \mathbin{;} x) \mathbin{;} \varphi)n \\ = & \qquad \text{hypothesis } (\gamma \mathbin{;} x)n = \delta n \end{array}$$

$$\begin{array}{ll} & (F\delta \mathbin{;} \varphi)n \\ = & \qquad \text{definition } \delta S \\ & (\delta S)n \end{array}$$

as desired. This completes the entire construction and proof.

7 Conclusion

We have given several simple examples, and at least one nontrivial one, of algebraic calculation in the framework of category theory. The calculations are quite smooth; there were few occasions where we had to interrupt a calculation, for establishing an auxiliary result or for introducing a (name for a) new morphism. Thanks to the systematisation of the notation and laws for the unique arrows brought forward by initiality, there is less or no need to draw or remember commutative diagrams for the inspiration or verification of a step in a calculation. Each step is easily verified, and there is ample opportunity for machine assistance in this respect. More importantly, the *construction* of required morphisms from others is performed as a calculation as well. There are several places where a morphism is constructed by beginning to prove the required property while, along the way, determining more and more of (an expression for) the morphism. Thus proof and construction go hand-in-hand, in an algebraic style.

All calculations can be interpreted in $\mathcal{S}et$ so that, actually, we have quite involved calculations with algorithms (functions). Calculations with algorithms working on more usual datatypes are explored extensively by, for instance, Malcolm [Mal90b] and Fokkinga [Fok92].

Acknowledgements. This note is heavily influenced by what other people have taught me in the past three years. In particular Roland Backhouse, Grant Malcolm, Lambert Meertens and Jaap van der Woude may recognise their ideas and methodological and notational suggestions. Also the comments of the referees helped to improve the presentation.

References

[AM75] M. Arbib and E. Manes. *Arrows, Structures and Functors — The Categorical Imperative*. Academic Press, 1975.

[Bac78] 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.

[Bir89] 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).

[BW90] M. Barr and C. Wells. *Category Theory for Computing Science.* Prentice Hall, 1990.

[CCM85] 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.

[DS90] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Springer Verlag, 1990.

[Fok91] M.M. Fokkinga. Datatype laws without signatures. Technical Report CS-R9133, CWI, Amsterdam, May 1991.

[Fok92] M.M. Fokkinga. *Law and Order in Algorithmics.* PhD thesis, University of Twente, dept Comp Sc, Enschede, Netherlands, 1992.

[Gas88] A.J.M. van Gasteren. *On the shape of Mathematical Arguments.* PhD thesis, Technical University of Eindhoven, 1988.

[Gol79] R. Goldblatt. *Topoi — the Categorical Analysis of Logic*, volume 98 of *Studies in Logic and the Foundations of mathematics.* North-Holland, 1979.

[GS89] J. Gray and A. Scedrov, editors. *Categories in Computer Science and Logic*, volume 92 of *Contempory mathematics.* 1989.

[Hag87a] T. Hagino. *Category Theoretic Approach to Data Types.* PhD thesis, University of Edinburgh, 1987.

[Hag87b] 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.

[Hoa89] 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).

[Hug90] J. Hughes. Why functional programming matters. In D.A. Turner, editor, *Research Topics in Functional Programming*, pages 17–42. Addison-Wesley, 1990.

[Mal90a] G. Malcolm. *Algebraic Data Types and Program Transformation.* PhD thesis, Groningen University, Netherlands, 1990.

[Mal90b] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14(2–3):255–280, September 1990.

[Mee86] 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.

[Mei92] E. Meijer. *Calculating Compilers*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1992.

[ML71] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, 1971.

[PAPR86] 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.

[Pat90] R. Paterson. Operators. In *Lecture Notes International Summerschool on Constructive Algorithmics*, September 1990. Organized by CWI Amsterdam, Utrecht University, University of Nijmegen, and held at Hollum (Ameland), Netherlands.

[Pie91] B.C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, Cambridge, Ma, 1991. Originally planned for Computing Surveys. Also Tech Report CMU-CS-90-113, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 125213.

[PPR87] D. Pitt, A. Poigné, and D. Rydeheard, editors. *Category Theory and Computer Science*, volume 283 of *Lect. Notes in Comp. Sc.* Springer Verlag, 1987.

[RB88] D.E. Rydeheard and R.M. Burstall. *Computational Category Theory*. Prentice Hall, 1988.

[Rey80] 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.