Categorical semantics for arrows

Bart Jacobs, Chris Heunen, Ichiro Hasuo
Institute for Computing and Information Sciences
Radboud University, Nijmegen, the Netherlands
(*e-mail:* {`b.jacobs,c.heunen,i.hasuo`}`@cs.ru.nl`)

Abstract

Arrows are an extension of the well-established notion of a monad in functional programming languages. This article presents several examples and constructions, and develops denotational semantics of arrows as monoids in categories of bifunctors $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{C}$. Observing similarities to monads – which are monoids in categories of endofunctors $\mathbf{C} \to \mathbf{C}$ – it then considers Eilenberg-Moore and Kleisli constructions for arrows. The latter yields Freyd categories, mathematically formulating the folklore claim "arrows are Freyd categories".

Contents

1 Introduction

The motivation to introduce the concept of an arrow comes from functional programming (Hughes, 2000; Paterson, 2001). It is intended as a uniform interface to certain types of computations, streamlining the infrastructure. This enables a high level of abstraction to uniformly capture for instance quantum computing (Vizzotto *et al.*, 2006+). It also facilitates language extensions like secure information flow (Li & Zdancewic, 2008): instead of building a domain-specific programming language from the ground up, it can be defined within normal Haskell using the arrow interface. After all, arrows provide an abstract interface supporting familiar programming constructs like composition, conditional branches and iteration. Haskell even incorporates convenient syntax to ease the use of such language extensions. The name "arrow" reflects the focus on the provided infrastructure, especially compositionality.[1]

Here is a more mathematical intuition. Monoids are probably the most fundamental mathematical structures used in computer science. The basic example $(A, ;, skip)$ is given by a set $A \in \mathbf{Set}$ of programs or actions, with sequential composition ; as binary operation, and an empty statement *skip* as neutral element for composition. Such a monoid A does not capture input and output. We may like to add it via parametrisation $A(X, Y)$, where X, Y are type variables. Since input is contravariant and output covariant, we may consider such an indexed monoid $A(-, +)$ as a bifunctor $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ for a suitable category $\mathbf{C}$ of types for input and output. But of course, we still want it to have a monoid structure for composition. Hence we are led to consider monoids in the functor category $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$. Our first main result – stemming from (Heunen & Jacobs, 2006) – is that such monoids are in fact arrows as introduced by Hughes.

A special case of the above is when there is only output, but no input: these singly indexed monoids are (categorical) monads. They correspond to the well-known notion of a monad in Haskell (Moggi, 1989; Wadler, 1992). Arrows are thus similar to monads in that they are monoids in suitable categories: namely in categories of endofunctors $\mathbf{C} \to \mathbf{C}$. Hence we are led to ask: what are the Eilenberg-Moore and Kleisli constructions – two very basic constructions on monads – for arrows? Our second main result – from (Jacobs & Hasuo, 2006) – is that the Kleisli construction for arrows corresponds to *Freyd categories* (Robinson & Power, 1997), and moreover the correspondence is isomorphic. Thus, to the folklore claim "arrows are Freyd categories" that we put in precise terms, we add the slogan "Freyd is Kleisli, for arrows".

These main results are streamlined versions of (Heunen & Jacobs, 2006) and (Jacobs & Hasuo, 2006). The current article proceeds as follows. In Section 2 we introduce the concepts of monads and arrows in Haskell more thoroughly, gradually moving towards a more mathematical mindset instead of a functional programming perspective. We also motivate why one can in fact achieve more with arrows than

[1] In a categorical context, the name is a bit unfortunate, however. We consistenly use "arrow" for the programming construct and "morphism" for the categorical notion.

with monads, and give settings where this is useful. Section 3 investigates, still in a somewhat discursive style, combinations of arrows. It leads up to a deconstruction into elementary parts of the particular program that motivated Hughes to use arrows in the first place (Swierstra & Duponcheel, 1996). The formal, categorical, analysis of arrows takes place in Section 4, culminating in our first main result mentioned above, Corollary 4.1. An example showing the elegance of this approach is discussed in Section 5, namely arrows facilitating bidirectional computation. Section 6 then considers algebra constructions for arrows, and contains the second main result, Theorem 6.2. We conclude in section 7. Appendix A contains a proof of a result used in Section 4 but only sketched there. Next, Appendix B considers a bicategorical characterisation of the notion of arrow that elegantly exemplifies its naturality, but is somewhat out of the scope of the main line of this article. Finally, Appendix C gives the missing details of Section 6.

2 Haskell examples

This section introduces arrows and their use in functional programming languages. We briefly consider monads first in subsection 2.1, since this construction from category theory historically paved the way for arrows (subsection 2.2). Subsection 2.3 then consider the advantages of arrows over monads.

2.1 Monads

A major reason for the initial reluctance to adopt functional programming languages is the need to pass state data around explicitly. Monadic programming provides an answer to this inconvenience (Moggi, 1989; Wadler, 1992). Through the use of a monad one can encapsulate the changes to the state data, the "side-effects", without explicitly carrying states around. monads can efficiently structure functional programs while improving genericity. This mechanism is even deemed important enough to be incorporated into Haskell syntax (Peyton Jones, 2003). A monad in Haskell is defined as a so-called type class:

$$\begin{array}{lll} \mathit{class} & \mathit{Monad}\ M\ \mathit{where} & \\ & \mathit{return} & ::\ X \to M\ X \\ & (\ggg\!=) & ::\ M\ X \to (X \to M\ Y) \to M\ Y \end{array}$$

To ensure the desired behaviour, the programmer herself should prove certain *monad laws* about the operations *return* and $\ggg\!=$ (pronounced *bind*). These boil down to the axioms that M be a monad, in the categorical sense. Using the formulation that is standard in the functional programming community, a categorical monad consist of a mapping $X \mapsto M(X)$ on types, together with "return" and "bind" functions

$$X \xrightarrow{\mathrm{rt}} MX, \qquad (X \to MY) \xrightarrow{\mathrm{bd}} (MX \to MY)$$

satisfying

$$\mathrm{bd}(f) \circ \mathrm{rt} = f, \qquad \mathrm{bd}(\mathrm{rt}) = \mathrm{id}, \qquad \mathrm{bd}(f) \circ \mathrm{bd}(g) = \mathrm{bd}(\mathrm{bd}(f) \circ g).$$

In categorical style one defines M to be a functor, with multiplication maps $\mu = \mathrm{bd}(\mathrm{id}_{MX}) : M^2X \to MX$ satisfying suitable laws. The above equations are more convenient for equational reasoning. Often one writes $u \ggg\!= f$ for $\mathrm{bd}(f)(u)$.

The most familiar monads are powerset, list, lift, state and distribution:

$$\begin{array}{lll}
\mathcal{P} & \mathrm{rt}(x) = \{x\} & \mathrm{bd}(f)(a) = \bigcup\{f(x) \mid x \in a\} \\
(-)^{\star} & \mathrm{rt}(x) = \langle x \rangle & \mathrm{bd}(f)(\langle x_1, \ldots, x_n \rangle) = f(x_1) \cdot \ldots \cdot f(x_n) \\
1 + (-) & \mathrm{rt}(x) = \mathrm{up}(x) & \mathrm{bd}(f)(v) = \begin{cases} \perp & \text{if } v = \perp \\ f(x) & \text{if } v = \mathrm{up}(x) \end{cases} \\
(- \times S)^S & \mathrm{rt}(x) = \lambda s \,.\, \langle x, s \rangle & \mathrm{bd}(f)(h) = \lambda s \,.\, f\big(\pi_1 h(s)\big)\big(\pi_2 h(s)\big) \\
\mathcal{D} & \mathrm{rt}(x) = \lambda y \,.\, \begin{cases} 1 \text{ if } x = y \\ 0 \text{ if } x \neq y \end{cases} & \mathrm{bd}(f)(\varphi) = \lambda y \,.\, \sum_x \varphi(x) \cdot f(x)(y)
\end{array}$$

In the latter case we write $\mathcal{D}$ for the 'subdistribution' monad $\mathcal{D}(X) = \{\varphi\colon X \to [0,1] \mid \mathrm{supp}(\varphi) \text{ is finite and } \sum_x \varphi(x) \leq 1\}$, where the support $\mathrm{supp}(\varphi)$ is the set of $x \in X$ with $\varphi(x) > 0$.

Monads are often considered with strength, *i.e.* come equipped with a suitable natural transformation $\mathrm{st} : M(X) \times Y \to M(X \times Y)$. For later reference, we use that in our present informal setting each functor M is strong, as its strength can be described explicitly as:

$$\mathrm{st}(u, y) = M\big(\lambda x \,.\, \langle x, y \rangle\big)(u). \tag{1}$$

It satisfies the following basic equations.

$$\begin{aligned}
M(f \times g) \circ \mathrm{st} &= \mathrm{st} \circ (M(f) \times g), \\
M(\pi_1) \circ \mathrm{st} &= \pi_1, \\
M(\alpha^{-1}) \circ \mathrm{st} &= \mathrm{st} \circ (\mathrm{st} \times \mathrm{id}) \circ \alpha^{-1},
\end{aligned}$$

where we use

$$\pi_i : X_1 \times X_2 \to X_i, \qquad\qquad \alpha : (X \times Y) \times Z \xrightarrow{\cong} X \times (Y \times Z),$$

for the familiar product maps fst, snd and assoc.

In other, non-set-theoretic settings one may have to require such strength maps explicitly. The monad operations interact appropriately with the above strength map, in the sense that the following equations hold.

$$\mathrm{st} \circ (\mathrm{rt} \times \mathrm{id}) = \mathrm{rt} \qquad \mathrm{st} \circ (\mathrm{bd}(f) \times g) = \mathrm{bd}(\mathrm{st} \circ f \times g) \circ \mathrm{st}.$$

In effect, monads are thus functional combinators. They enable the combination of functions very generally, without many assumptions on the precise functions to combine. However, these restrictions are severe enough to exclude certain classes of libraries from implementation with a monadic interface.

2.2 Arrows

Arrows are even more general functional combinators, and can be seen as a generalisation of monads (Hughes, 2000; Hughes, 2005). An arrow in Haskell is a type class of the form

$$\begin{array}{lll} \mathit{class} & \mathit{Arrow}\ A\ \mathit{where} & \\ & \mathit{arr} & :: \quad (X \to Y) \to A\ X\ Y \\ & (\ggg) & :: \quad A\ X\ Y \to A\ Y\ Z \to A\ X\ Z \\ & \mathit{first} & :: \quad A\ X\ Y \to A\ (X,Z)\ (Y,Z) \end{array}$$

where X, Z in Haskell denotes the Cartesian product type $X \times Y$. Analogous to monads, an arrow must furthermore satisfy the following *arrow laws*, the proof of which is up to the programmer.

$$(a \ggg b) \ggg c = a \ggg (b \ggg c), \tag{2}$$

$$\mathit{arr}\ (g \circ f) = \mathit{arr}\ f \ggg \mathit{arr}\ g, \tag{3}$$

$$\mathit{arr}\ \mathrm{id} \ggg a = a = a \ggg \mathit{arr}\ \mathrm{id}, \tag{4}$$

$$\mathit{first}\ a \ggg \mathit{arr}\ \pi_1 = \mathit{arr}\ \pi_1 \ggg a, \tag{5}$$

$$\mathit{first}\ a \ggg \mathit{arr}\ (\mathrm{id} \times f) = \mathit{arr}\ (\mathrm{id} \times f) \ggg \mathit{first}\ a, \tag{6}$$

$$\mathit{first}\ (\mathit{first}\ a) \ggg \mathit{arr}\ \alpha = \mathit{arr}\ \alpha \ggg \mathit{first}\ a, \tag{7}$$

$$\mathit{first}\ (\mathit{arr}\ f) = \mathit{arr}\ (f \times \mathrm{id}), \tag{8}$$

$$\mathit{first}\ (a \ggg b) = \mathit{first}\ a \ggg \mathit{first}\ b, \tag{9}$$

In fact, as Section 6.1 shows, less structure than Cartesian products suffices, eliminating the need for projections π_i in the above arrow laws. Sometimes, $\mathit{arr}(\mathrm{id} \times f)$ is written as $\mathit{second}(\mathit{arr}(f))$, where

$$\mathit{second}(a) \quad = \quad \mathit{arr}(\gamma) \ggg \mathit{first}(a) \ggg \mathit{arr}(\gamma),$$

and $\gamma : X \times Y \xrightarrow{\cong} Y \times X$ is the well-known swap-map. The arrow laws (2)–(9) are sometimes given names (Paterson, 2003). Especially noteworthy are the names "exchange" for (6) and "extension" for (8).

Example of arrows will be given in Section 3.

2.3 Monads versus arrows

This article is concerned with a categorical understanding of this notion of arrow. At this stage we shall reveal some of the structure involved, but are deliberately a bit vague about the general setting in which we are working. In doing so we move to a more mathematical notation, for instance writing $A(X, Y)$ for $A\,X\,Y$ in functional style.

It is not hard to show that an arrow is "bifunctorial" (Lemma 4.1). This means that for $f : X' \to X$ and $g : Y \to Y'$ one also has a map $A(X, Y) \to A(X', Y')$. The maps $\mathit{arr} : Y^X \to A(X, Y)$ then form natural transformations (Lemma 4.2). Even more, composition can also be seen as a natural transformation $A \otimes A \to A$, for a suitable tensor product $\otimes$ of bifunctors (Proposition 4.2). In this way one can describe the triple $(A, \mathit{arr}, \ggg)$ as a monoid in a category of bifunctors. Here we shall not need these details yet. But in the remainder of this section we shall introduce arrows as bifunctors of the form $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$.

Here is a first trivial example. Let (P, m, e) be a monoid, consisting of an associative operation $m : P \times P \to P$ with two-sided unit $e : P$. It yields probably the most elementary example of an arrow, namely a constant one. We shall also write it as P, formally as functor in $P(X, Y) = P$, with operations:

$$arr(f) = e, \qquad a \ggg b = m(a, b) \qquad first(a) = a.$$

Standard examples of monoids P are the singleton type 1 (with trivial operations), the type $2 = \{0, 1\}$ of truth values or Booleans (with either conjunctions $\top, \wedge$ or disjunctions $\bot, \vee$), or the type $X^\star$ of lists of an arbitrary type X (with the empty list $\langle\rangle$ and concatenation $\cdot$).

Every monad $(M, \mathrm{rt}, \mathrm{bd})$ with a strength gives rise to an arrow $\overline{M}$ by

$$\overline{M}(X, Y) = M(Y)^X, \tag{10}$$

with obvious operations (see *e.g.* Hughes, 2000) – strength is used to provide the operation $first$.

Dual to a monad, a comonad is given by a mapping $X \mapsto N(X)$ with "coreturn" and "cobind" operations $\mathrm{crt} : NX \to X$ and $\mathrm{cbd} : (NX \to Y) \longrightarrow (NX \to NY)$ satisfying

$$\mathrm{crt} \circ \mathrm{cbd}(f) = f, \qquad \mathrm{cbd}(\mathrm{crt}) = \mathrm{id}, \qquad \mathrm{cbd}(f) \circ \mathrm{cbd}(g) = \mathrm{cbd}(f \circ \mathrm{cbd}(g)).$$

It gives rise to an arrow by $(X, Y) \mapsto Y^{N(X)}$ – no strength is needed.

Comonads are less well-known, but are fundamental structures for handling contexts (among other things), in which the "counit" $\varepsilon = \mathrm{crt} : NX \to X$ is used for weakening, and the "comultiplication" $\delta = \mathrm{cbd}(\mathrm{id}_{NX}) : NX \to N^2X$ for contraction (Jacobs, 1999). The following diagram presents the main comonads $X \mapsto \cdots$ for handling streams with discrete time input (Uustalu & Vene, 2005).

$$\begin{array}{ccccc} X^\star \times X & \xleftarrow[\text{no future}]{\text{causality}} & X^{\mathbb{N}} \times \mathbb{N} & \xrightarrow[\text{no past}]{\text{anti-causality}} & X^{\mathbb{N}} \\ (\langle \alpha(0), \ldots, \alpha(n-1) \rangle, \alpha(n)) & \longleftarrow\!\!\shortmid & (\alpha, n) & \longmapsto & \lambda m \,.\, \alpha(n+m) \end{array} \tag{11}$$

The intuition for a pair $(\alpha, n) \in X^{\mathbb{N}} \times \mathbb{N}$ is that n represents the present stage in the stream $\alpha = \langle \alpha(0), \alpha(1), \ldots, \alpha(n-1), \alpha(n), \alpha(n+1), \ldots \rangle$, where everything before n is past input, and everything after n is future input. The two morphisms in the previous diagram are homomorphisms of comonads, commuting with the relevant comonad/context structure. There is a similar real-time analogue.

A strong monad M and a comonad N can also be combined to form arrows. As illustrated for instance in (Uustalu & Vene, 2005; Heunen & Jacobs, 2006), this happens via a so-called distributive law $NM \Rightarrow MN$ that commutes with the (co)monad operations. Then one can define an arrow $\overline{(M, N)}$ via

$$\overline{(M, N)}(X, Y) = M(Y)^{N(X)}. \tag{12}$$

It combines the previous two constructions with monads and comonads separately.

This mapping $(X, Y) \mapsto M(Y)^{N(X)}$ leads to an appealing picture of an arrow in which the monad M is used for structuring the outputs and the comonad N for the inputs. But arrows are more general than this. For instance, if we wish to do "non-deterministic dataflow" we may consider at first maps of the form

$$X^{\mathbb{N}} \times \mathbb{N} \longrightarrow \mathcal{P}(Y), \tag{13}$$

with the comonad on the left-hand side structuring the input of streams, and the monad on the right-hand-side producing non-deterministic output. However, this requires a distributive law of the form

$$\mathcal{P}(X)^{\mathbb{N}} \times \mathbb{N} \longrightarrow \mathcal{P}(X^{\mathbb{N}} \times \mathbb{N}).$$

While it is possible to construct such a function – for instance the power law from (Jacobs, 2006) – it does not commute with the comonad structure. As a result, composition is not associative.

The way out is to realise that co-Kleisli maps $X^{\mathbb{N}} \times \mathbb{N} \to Y$ correspond to maps $X^{\mathbb{N}} \to Y^{\mathbb{N}}$ via Currying. But then non-determinism can be introduced easily into dataflow, namely by looking at maps

$$X^{\mathbb{N}} \longrightarrow \mathcal{P}(Y^{\mathbb{N}}) \tag{14}$$

instead of maps (13). The corresponding assignment $(X, Y) \mapsto \mathcal{P}(Y^{\mathbb{N}})^{(X^{\mathbb{N}})}$ indeed forms an arrow – with associative composition. It is however not of the form $(X, Y) \mapsto M(Y)^{N(X)}$. Arrows thus have more to offer than monad-comonad combinations. As an aside: it is not so clear how to combine the other comonads in (11) with non-determinism.

3 Arrow constructions and examples

This section continues in the discursive style of the previous one. It introduces several elementary ways to combine arrows, and use these constructions to obtain some well-known examples. The first construction is obvious, but useful. Its proof is straightforward and left to the reader.

Lemma 3.1
Let $(A_1, arr_1, \ggg_1)$ and $(A_2, arr_2, \ggg_2)$ be arrows. Then so is their product $A = A_1 \times A_2$, described by

$$A(X, Y) = A_1(X, Y) \times A_2(X, Y)$$

with operations

$$\begin{aligned} arr(f) &= \langle arr_1(f), arr_2(f)\rangle \\ \langle a_1, a_2\rangle \ggg \langle b_1, b_2\rangle &= \langle a_1 \ggg_1 b_1, a_2 \ggg_2 b_2\rangle \\ first(\langle a, b\rangle) &= \langle first_1(a), first_2(b)\rangle. \quad \square \end{aligned}$$

The next result now follows from the observation in the previous section that each monoid forms a (constant) arrow. The result is mentioned explicitly because it will be used later in this form, in Example 3.1.

Corollary 3.1

Let $(A, arr, \ggg)$ be an arrow, and (P, m, e) be a monoid. Then $A' = P \times A$, given by

$$A'(X,Y) = \big(P \times A\big)(X,Y) = P \times \big(A(X,Y)\big),$$

is again an arrow, with the following operations.

$$\begin{aligned} arr'(f) &= \langle e, arr(f)\rangle \\ \langle x, a\rangle \ggg' \langle y, b\rangle &= \langle m(x,y), a \ggg b\rangle \\ first'(\langle x, a\rangle) &= \langle x, first(a)\rangle. \quad \square \end{aligned}$$

For the next result we consider functors F that *preserve products*. This means that the obvious maps

$$F(X \times Y) \xrightarrow{\langle F(\pi_1), F(\pi_2)\rangle} F(X) \times F(Y)$$

are isomorphisms. In that case we shall write $\beta = \beta_{X,Y} : F(X) \times F(Y) \to F(X \times Y)$ for the inverse.

Lemma 3.2
Let $(A, arr, \ggg)$ be an arrow, and F be a product preserving functor. Defining

$$A_F(X,Y) = A(F(X), F(Y)),$$

yields a new arrow A_F with the following operations.

$$\begin{aligned} arr'(f) &= arr(F(f)) \\ a \ggg' b &= a \ggg b \\ first'(a) &= arr(\langle F(\pi_1), F(\pi_2)\rangle) \ggg first(a) \ggg arr(\beta). \end{aligned}$$

Proof
Checking the relevant equations is not hard. For instance:

$$\begin{aligned} & first'(a \ggg b) \\ &\quad = \quad arr(\langle F(\pi_1), F(\pi_2)\rangle) \ggg first(a \ggg b) \ggg arr(\beta) \\ &\quad \overset{(9)}{=} \quad arr(\langle F(\pi_1), F(\pi_2)\rangle) \ggg first(a) \ggg first(b) \ggg arr(\beta) \\ &\quad \overset{(4)}{=} \quad arr(\langle F(\pi_1), F(\pi_2)\rangle) \ggg first(a) \ggg arr(\langle F(\pi_1), F(\pi_2)\rangle \circ \beta) \\ &\qquad\qquad \ggg first(b) \ggg arr(\beta) \\ &\quad \overset{(3)}{=} \quad arr(\langle F(\pi_1), F(\pi_2)\rangle) \ggg first(a) \ggg arr(\beta) \\ &\qquad\qquad \ggg arr(\langle F(\pi_1), F(\pi_2)\rangle) \ggg first(b) \ggg arr(\beta) \\ &\quad = \quad first'(a) \ggg' first'(b). \quad \square \end{aligned}$$

Lemma 3.3
Let $(A, arr, \ggg)$ be an arrow and S an arbitrary type. The definition

$$A_{S\times}(X,Y) = A(S \times X, S \times Y)$$

again yields an arrow, with corresponding structure:

$$\begin{aligned} arr_{S\times}(f) &= arr(\mathrm{id}_S \times f) \\ a \ggg_{S\times} b &= a \ggg b \\ first_{S\times}(a) &= arr(\alpha^{-1}) \ggg first(a) \ggg arr(\alpha) \end{aligned}$$

where α is the associativity isomorphism for products from Subsection 2.1. $\square$

This particular construction $A_{S\times}$ occurs, in a slightly different formulation, already in (Hughes, 2000, §§9) where it is introduced via a 'state functor'. A similar construction $(X, Y) \mapsto A(X, S)^{A(Y,S)}$ is defined there for special arrows with suitable apply operations $A(A(X, Y) \times X, Y)$.

At this stage we can already see how one of the motivating examples for the notion of arrow can be obtained from the previous constructions.

Example 3.1
In (Hughes, 2000, §§4.2) an arrow *SD* is introduced to describe a special parser defined by (Swierstra & Duponcheel, 1996). This arrow can be described as

$$SD(X, Y) = (2 \times S^\star) \times \big(1 + S^\star \times Y\big)^{(S^\star \times X)}. \tag{15}$$

We show that this arrow *SD* can be obtained by successive application of the constructions in this section.

First, the set $2 \times S^\star$ – with $2 = \{0, 1\}$ – is used as monoid, not with the standard structure, but with unit and composition given by:

$$\begin{aligned} e &= (1, \langle\rangle) \\ m((b, \sigma), (c, \tau)) &= (b \wedge c, \sigma \cdot (if\ b = 1\ then\ \tau\ else\ \langle\rangle)). \end{aligned}$$

It is not hard to see that this yields a monoid. Corollary 3.1 then tells that (15) is an arrow if the rightmost part $(X, Y) \mapsto \big(1 + S^\star \times Y\big)^{(S^\star \times X)}$ is. Using the lift monad $1+(-)$ we get an arrow $(X, Y) \mapsto (1+Y)^X$, as shown in subsection 2.3. By applying Corollary 3.3 with set $S^\star$ we obtain the rightmost part, as required.

When we go into the details of these constructions we can also reconstruct the associated operations of the arrow (15) as follows.

$$\begin{aligned}
&arr(f) \\
&\quad = \ \langle\, (1, \langle\rangle),\ \lambda(s, x) \in S^\star \times X \,.\, \mathrm{up}(s, f(x))\,\rangle \\
&\langle (b, \sigma), f)\rangle \ggg \langle (c, \tau), g\rangle \\
&\quad = \ \langle\, (b \wedge c, \sigma \cdot (if\ b = 1\ then\ \tau\ else\ \langle\rangle)), \\
&\qquad\qquad \lambda(s, x) \in S^\star \times X \,.\, \begin{cases} \bot & \text{if } f(s, x) = \bot \\ g(t, y) & \text{if } f(s, x) = \mathrm{up}(t, y)\ \rangle \end{cases} \\
&first(\langle (b, \sigma), f\rangle) \\
&\quad = \ \langle\, (b, \sigma), \\
&\qquad\qquad \lambda(s, (x, y)) \in S^\star \times (X \times Y) \,.\, \begin{cases} \bot & \text{if } f(s, x) = \bot \\ \mathrm{up}(t, (z, y)) & \text{if } f(s, x) = \mathrm{up}(t, z)\ \rangle. \end{cases}
\end{aligned}$$

These operations are precisely as described (in Haskell notation) in (Hughes, 2000, §§4.2).

Example 3.2
Quantum computing (Vizzotto *et al.*, 2006+) can be modelled within a functional programming language. The states of a quantum program are so-called *density matrices*, that we can understand as elements of the monad application $\mathcal{D}(X)^X$ to some set X. These states evolve into each other by *superoperators*, which can be

modelled as arrows $(X,Y) \mapsto \mathcal{D}(Y \times Y)^{(X \times X)}$. The previous lemmas also enable us to show that this quantum computation arrow is indeed an arrow, by decomposing it into elementary parts, without checking the arrow laws by hand.

First, recall that the mapping $(X,Y) \mapsto \mathcal{D}(Y)^X$ yields an arrow, induced by the distribution monad $\mathcal{D}$. Next, notice that the diagonal functor $X \mapsto X \times X$ preserves products, so that the mapping $(X,Y) \mapsto (Y \times Y)^{(X \times X)}$ yields an arrow, with

$$first(a) = \lambda((x,z),(x',z')) \in (X \times Z) \times (X \times Z) \,.\, ((\pi_1 a(x,x'),z),(\pi_2 a(x,x'),z'))$$

for given $a : X \times X \to Y \times Y$,

Thus, according to Lemma 3.2, the mapping $(X,Y) \mapsto \mathcal{D}(Y \times Y)^{(X \times X)}$ is an arrow. If we follow through the construction, we obtain the following arrow operations.

$$\begin{array}{rcl}
arr(f) & = & \mathrm{rt} \circ (f \times f) \\
& = & \lambda(x,x') \,.\, \lambda(y,y') \,.\, \begin{cases} 1 & \text{if } f(x) = y \wedge f(x') = y' \\ 0 & \text{otherwise} \end{cases} \\
a \ggg b & = & \mathrm{bd}(b) \circ a \\
& = & \lambda(x,x') \,.\, \lambda(z,z') \,.\, \displaystyle\sum_{(y,y')} a(x,x')(y,y') \cdot b(y,y')(z,z') \\
first(a) & = & \mathcal{D}(\langle \pi_1 \times \pi_1, \pi_2 \times \pi_2 \rangle) \circ \mathrm{st} \circ (a \times \mathrm{id}) \circ \langle \pi_1 \times \pi_1, \pi_2 \times \pi_2 \rangle \\
& = & \lambda((x,z_1),(x',z_1')) \,.\, \lambda((y,z_2),(y',z_2')) \,.\, \begin{cases} a(x,x')(y,y') & \text{if } z_1 = z_2 \\ & \text{and } z_1' = z_2', \\ 0 & \text{otherwise.} \end{cases}
\end{array}$$

These indeed coincide exactly with the ones given in (Vizzotto *et al.*, 2006+).

4 Categorical formulation

In this section we shall move towards a categorical formulation of the notion of arrow. We shall do so by first analysing the structure in a Haskell-like setting. We denote by **HT** the category with Haskell types as objects. A morphism $\sigma \to \tau$ in this category is a Haskell function $f = \lambda x : \sigma \,.\, f(x) : \tau$ taking input in σ to output in τ. Composition of such maps is performed by substitution. Essentially, this is a Cartesian closed category of types and terms, but for the fact that some functions do not terminate, much like a lambda calculus. Of course there is much more structure (like general recursion) in Haskell than the types with type variables and terms, like in system F. Below we shall analyse the behaviour of Haskell arrows as bifunctors on **HT**, leading to a more general definition of an arrow over any category **C**.

4.1 Analysing arrow behaviour categorically

First and foremost, let us show that a Haskell arrow is indeed bifunctorial.

Lemma 4.1
The operation $A(-,-)$ extends to a functor $\mathbf{HT}^{\mathrm{op}} \times \mathbf{HT} \to \mathbf{Set}$ by

$$(X,Y) \mapsto \{a : A(X,Y) \mid a \text{ closed term}\},$$

whose action $A(f, g) : A(X, Y) \to A(X', Y')$ on maps $f : X' \to X$ and $g : Y \to Y'$ is given by

$$A(f, g) = \lambda a \,.\, arr(f) \ggg a \ggg arr(g).$$

Proof
Using equations (2), (3) and (4) one easily derives the functorial properties for identity, $A(\mathrm{id}, \mathrm{id}) = \mathrm{id}$, and composition, $A(f \circ f', g' \circ g) = A(f', g') \circ A(f, g)$. $\square$

We now examine the arrow operations arr and $first$ in the light of the bifunctoriality of A.

Lemma 4.2
The maps $arr : \mathbf{HT}(X, Y) \to A(X, Y)$ form a natural transformation $\mathbf{HT}(-, +) \Rightarrow A(-, +)$ from exponents to arrows, where $\mathbf{HT}(-, +)$ is the homset functor.

Similarly, the maps $first : A(X, Y) \to A(X \times Z, Y \times Z)$ are natural in X and Y. This may be formulated as: $first$ yields a natural transformation $\langle first \rangle$ from A to the functor $A\!\times$ given by $(X, Y) \mapsto \prod_Z A(X \times Z, Y \times Z)$. Of course, this functor $A\!\times$ only makes sense in a small category with arbitrary (set-indexed) products Π.

Proof
For maps $f : X' \to X$, $g : Y \to Y'$ in $\mathbf{HT}$ and $h : \mathbf{HT}(X, Y)$ we have

$$\begin{aligned}
\big(A(f, g) \circ arr\big)(h) &= arr(f) \ggg arr(h) \ggg arr(g) \\
&\overset{(3)}{=} arr(g \circ h \circ f) = arr(g^f(h)) = \big(arr \circ g^f\big)(h),
\end{aligned}$$

and

$$\begin{aligned}
\big(A\!\times(f, g) \circ \langle first \rangle\big)(a) &= \langle A(f \times \mathrm{id}, g \times \mathrm{id}) \circ \pi_Z \rangle_Z (\langle first(a) \rangle) \\
&= \langle A(f \times \mathrm{id}, g \times \mathrm{id})(first(a)) \rangle \\
&= \langle arr(f \times \mathrm{id}) \ggg first(a) \ggg arr(g \times \mathrm{id}) \rangle \\
&\overset{(8)}{=} \langle first(arr(f)) \ggg first(a) \ggg first(arr(g)) \rangle \\
&\overset{(9)}{=} \langle first(arr(f) \ggg a \ggg arr(g)) \rangle \\
&= \langle first(A(f, g)(a)) \rangle \\
&= \big(\langle first \rangle \circ A(f, g)\big)(a). \quad \square
\end{aligned}$$

The next lemma shows that the maps $\ggg : A(X, P) \times A(P, Y) \to A(X, Y)$ are natural in X and Y, just like the maps arr and $first$ in the previous lemma. In the parameter P they are what is called *dinatural* (Mac Lane, 1971, Section IX.4). This means that for each map $f : P \to Q$ the following diagram commutes.

$$\begin{tikzcd}
& A(X,P) \times A(P,Y) \arrow[r, "\ggg"] & A(X,Y) \arrow[dr, equal] & \\
A(X,P) \times A(Q,Y) \arrow[ur, "\mathrm{id} \times A(f,\mathrm{id})"] \arrow[dr, "A(\mathrm{id},f) \times \mathrm{id}"'] & & & A(X,Y) \\
& A(X,Q) \times A(Q,Y) \arrow[r, "\ggg"'] & A(X,Y) \arrow[ur, equal] &
\end{tikzcd}$$

Lemma 4.3
The maps $\ggg : A(X,P) \times A(P,Y) \to A(X,Y)$ are natural in X and Y, and dinatural in P.

Proof
Naturality is trivial. As for dinaturality, for $a : A(X,P)$ and $b : A(Q,Y)$, we have

$$\begin{aligned}\big(\ggg \circ (\mathrm{id} \times A(f,\mathrm{id}))\big)(a,b) &= a \ggg A(f,\mathrm{id})(b) \\ &= a \ggg arr(f) \ggg b \\ &= A(\mathrm{id},f)(a) \ggg b \\ &= \big(\ggg \circ (A(\mathrm{id},f) \times \mathrm{id})\big)(a,b). \quad \square\end{aligned}$$

Intuitively, dinaturality in P signifies that $\ggg$ is parametric in its middle argument type, and that this middle parameter is auxiliary; it could just have well been another one, as long as it is the same across the second argument of the first factor, and the first argument of the second.

4.2 Monoidal structure in the ambient category

Extending from the category **HT** of (Haskell) types and terms, we would like to define an arrow over any suitable category $\mathbf{C}$ as a monoid in the functor category $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$ of bifunctors that carries an internal strength. However, to do so we need to ensure that the ambient category, $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$, has monoidal structure. The most elegant way to achieve this is to employ the notion of (parametrized) coends, see Appendix A. This approach generalises to the $\mathbf{V}$-enriched situation, when an arrow is a suitable bifunctor $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$. Such enrichment is necessary if we are to consider (instead of **HT**) a categorical model of Haskell which is most probably **Cpo**-enriched. At this stage we shall present the construction for the reasonably concrete case where $\mathbf{V} = \mathbf{Set}$, mostly to give some intuition about the monoidal structure.

Proposition 4.1
Let $\mathbf{C}$ be a small category. Then the category $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$ of **Set**-valued bifunctors has a monoidal structure with unit I and tensor product $\otimes$.

Proof
The naturality of $\mathbf{HT}(-,+) \Rightarrow A(-,+)$ observed in Lemma 4.2 suggests that the (internal) homfunctor could serve as the unit of the intended monoidal structure on $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$. Thus we define $I : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ to be $\mathrm{Hom}_{\mathbf{C}}$; explicitly, $I(X,Y) = \mathbf{C}(X,Y)$ and $I(f,g) = g^f = \lambda h \,.\, g \circ h \circ f$. This requires $\mathbf{C}$ to be locally small.

The main idea now is to let the monoidal product of two bifunctors $A, B : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ be the smallest type containing all bifunctors that behave dinaturally in the middle parameter. More explicitly, composition $\ggg$ is a collection of morphisms

$$A(X,P) \times A(P,Y) \xrightarrow{\ggg} A(X,Y),$$

which can be combined, using the (arbitrary set-indexed) coproduct in **Set**, into one natural transformation with the following component at $X, Y \in \mathbf{C}$.

$$\left(\coprod_{P\in\mathbf{C}} A(X,P) \times A(P,Y)\right) \xrightarrow{\ggg} A(X,Y)$$

This requires $\mathbf{C}$ to have a (small) set of objects. We take the dinaturality of Lemma 4.3 into account by defining the components of the monoidal product $A \otimes B$ as the coequalizer c

$$\left(\coprod_{P,Q\in\mathbf{C}} \begin{array}{l} A(X,P) \\ \times\mathbf{C}(P,Q) \\ \times B(P,Y) \end{array}\right) \underset{d_2}{\overset{d_1}{\rightrightarrows}} \left(\coprod_{P\in\mathbf{C}} \begin{array}{l} A(X,P) \\ \times B(P,Y) \end{array}\right) \overset{c}{\dashrightarrow} (A\otimes B)(X,Y)$$

of (obvious cotuples of) the morphisms (in **Set**)

$$\begin{aligned} d_1 &= \lambda(a,f)\,.\,A(\mathrm{id},f)(a) : A(X,P)\times\mathbf{C}(P,Q) \to A(X,Q), \\ d_2 &= \lambda(f,b)\,.\,B(f,\mathrm{id})(b) : \mathbf{C}(P,Q)\times B(Q,Y) \to B(P,Y), \end{aligned}$$

for all $P, Q \in \mathbf{C}$. The composition maps $\ggg$ then reappear as the components of the unique $A \otimes A \Rightarrow A$ from the coequalizer. $\square$

Remark

The situation sketched in the previous proposition and proof is that of profunctors, which are also known as distributors or bimodules (Bénabou, 2000). Profunctors and natural transformations form a bicategory **Prof**, which is a well-studied generalisation of the category of sets and relations. The monoidal structure of **Prof** (as described above) is well-known. The basic idea is that composition of profunctors, and hence the tensor product in the above proposition, can also be written in terms of standard functor composition using left Kan extension along the Yoneda embedding. See (Day, 1970) for the original account, or (Borceux, 1994, Section 7.8) for a modern record.

The previous lemma puts us in a position to make precise our intuition that arrow laws (2)–(4) resemble monoid equations.

Proposition 4.2

An instantiation of the Haskell arrow class $(A\otimes A) \xrightarrow{\ggg} A \xleftarrow{arr} I$ satisfying (2)–(4) is a monoid in the category $\mathbf{Cat}(\mathbf{HT}^{\mathrm{op}}\times\mathbf{HT}, \mathbf{Set})$ of bifunctors $\mathbf{HT}^{\mathrm{op}}\times\mathbf{HT} \to \mathbf{Set}$.

Proof

We have to check that the monoid equations hold for the span $(A\otimes A) \xrightarrow{\ggg} A \xleftarrow{arr} I$. Here we exhibit one of the equations, namely

$$\begin{tikzcd} A \arrow[r, "\rho^{-1}"] \arrow[drr, equal] & A\otimes I \arrow[r, "\mathrm{id}\otimes arr"] & A\otimes A \arrow[d, "\ggg"] \\ & & A, \end{tikzcd}$$

which for $a : A(X,Y)$ becomes

$$\begin{tikzcd} a \arrow[r, mapsto] & (a, \mathrm{id}) \arrow[r, mapsto] & (a, arr(\mathrm{id})) \arrow[d, mapsto] \\ & & a \ggg arr(\mathrm{id}). \end{tikzcd}$$

Hence commutation of this diagram amounts to arrow law (4), which states that $a \ggg arr(\mathrm{id}) = a$. $\square$

Remark

Although the proof of Proposition 4.1 requires a restriction to small categories, we will often relax this to locally small categories. We are only after $A \otimes A$ anyway, and indeed, in the construction of $A \otimes A$ above we used a large coproduct for clarity, where we could have formulated the composition operation $\ggg$ of A via collections of maps $A(X,P) \times A(P,Y) \to A(X,Y)$ that are natural in X, Y, dinatural in P and satisfy the arrow equations (2)–(9).

In this way one could include domain theoretic models that are standardly used for Haskell semantics.

4.3 Internal strength

Now that we have seen that arrow laws (2)–(4) correspond to the monoid equations on the semantical side, we investigate the remaining laws (5)–(9) concerning $first$ in more detail.

Recall that a monad $T : \mathbf{C} \to \mathbf{C}$ on a monoidal category $\mathbf{C}$ is called *strong* when there is a natural transformation "strength" with components $\mathrm{st}_{X,Y} : T(X) \otimes Y \to T(X \otimes Y)$ that satisfies suitable coherence conditions. This section shows that the availability of the function $first$ is equivalent to an analogous form of strength for bifunctors, that we call internal strength. Its emergence is motivated in Appendix B.

Definition 4.1

Let $\mathbf{C}$ be a category with finite products. The carrier $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ of a monoid $(A, \ggg, arr)$ in $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$ is said to carry an *internal strength* natural transformation with components $\mathrm{ist}_{X,Y} : A(X,Y) \to A(X, Y \times X)$ if these satisfy

$$\mathrm{ist}(arr(f)) = arr(\langle f, \mathrm{id}\rangle), \tag{16}$$

$$\mathrm{ist}(a) \ggg arr(\pi_1) = a, \tag{17}$$

$$\mathrm{ist}(a \ggg b) = \mathrm{ist}(a) \ggg \mathrm{ist}(arr(\pi_1) \ggg b) \ggg arr(\mathrm{id} \times \pi_2), \tag{18}$$

$$\mathrm{ist}(\mathrm{ist}(a)) = \mathrm{ist}(a) \ggg arr(\langle \mathrm{id}, \pi_2\rangle). \tag{19}$$

Using the techniques of Appendix A this can again be extended to bifunctors $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$ for a category $\mathbf{C}$ with finite products and a suitable category $\mathbf{V}$.

The following proposition shows that having internal strength is in fact equivalent to having a $first$ operation for arrows – as originally introduced by Hughes.

Proposition 4.3

Let $(A, \ggg, arr)$ be an instantiation of the Haskell arrow class satisfying (2)-(4). The maps $first : A(X,Y) \to A(X \times Z, Y \times Z)$ satisfying equations (5)–(9) correspond to maps $\mathrm{ist} : A(X,Y) \to A(X, Y \times X)$ which are natural in Y and dinatural in X, and satisfy (16)–(19).

Proof

The proof of the equivalence of $first$ and ist involves many basic calculations, of which we only present a few exemplaric cases.

Given maps $first$ satisfying (5)–(9), define internal strength on $a : A(X,Y)$ as

$$\mathrm{ist}(a) = arr(\Delta) \ggg first(a),$$

where $\Delta = \langle \mathrm{id}, \mathrm{id} \rangle$. One then checks naturality in Y, dinaturality in X, and (16)–(19). The (di)naturality equations can be formulated as:

$$\mathrm{ist}(a) \ggg arr(g \times \mathrm{id}) = \mathrm{ist}(a \ggg arr(g)) \tag{20}$$

$$arr(f) \ggg \mathrm{ist}(a) = \mathrm{ist}(arr(f) \ggg a) \ggg arr(\mathrm{id} \times f). \tag{21}$$

By way of illustration we check equation (17):

$$\begin{aligned}
\mathrm{ist}(a) \ggg arr(\pi_1) &= arr(\Delta) \ggg first(a) \ggg arr(\pi_1) \\
&\overset{(5)}{=} arr(\Delta) \ggg arr(\pi_1) \ggg a \\
&\overset{(3)}{=} arr(\pi_1 \circ \Delta) \ggg a \\
&= arr(\mathrm{id}) \ggg a \\
&\overset{(4)}{=} a.
\end{aligned}$$

Conversely, given internal strength maps ist satisfying (16)–(19), define:

$$first(a) = \mathrm{ist}(arr(\pi_1) \ggg a) \ggg arr(\mathrm{id} \times \pi_2),$$

where $\pi_1 : X \times Z \to X$ and $\mathrm{id} \times \pi_2 : Y \times (X \times Z) \to Y \times Z$. This yields a natural operation, in the sense that:

$$arr(f \times \mathrm{id}) \ggg first(a) \ggg arr(g \times \mathrm{id}) = first(f \ggg a \ggg g).$$

We shall prove equation (9) in detail, and leave the rest to the interested reader.

$$\begin{aligned}
&first(a) \ggg first(b) \\
&\quad = \quad \mathrm{ist}(arr(\pi_1) \ggg a) \ggg arr(\mathrm{id} \times \pi_2) \ggg \mathrm{ist}(arr(\pi_1) \ggg b) \\
&\qquad\qquad \ggg arr(\mathrm{id} \times \pi_2) \\
&\quad \overset{(\mathrm{dinat})}{=} \quad \mathrm{ist}(arr(\pi_1) \ggg a) \ggg \mathrm{ist}(arr(\mathrm{id} \times \pi_2) \ggg arr(\pi_1) \ggg b) \\
&\qquad\qquad \ggg arr(\mathrm{id} \times (\mathrm{id} \times \pi_2)) \ggg arr(\mathrm{id} \times \pi_2) \\
&\quad \overset{(3)}{=} \quad \mathrm{ist}(arr(\pi_1) \ggg a) \ggg \mathrm{ist}(arr(\pi_1) \ggg b) \\
&\qquad\qquad \ggg arr(\mathrm{id} \times \pi_2) \ggg arr(\mathrm{id} \times \pi_2) \\
&\quad \overset{(18)}{=} \quad \mathrm{ist}(arr(\pi_1) \ggg a \ggg b) \ggg arr(\mathrm{id} \times \pi_2) \\
&\quad = \quad first\ (a \ggg b).
\end{aligned}$$

□

The alternative formulation in terms of internal strength ist in the previous

Proposition is convenient because its (di)naturality is clearly described, and it has only two parameters, whereas $first$ has three.

4.4 The categorical definition

After the preparations of section 4.2 we know that an arrow A satisfying arrow laws (2)–(4) is precisely a monoid in the category of bifunctors $\mathbf{HT}^{\mathrm{op}} \times \mathbf{HT} \to \mathbf{Set}$. Furthermore, section 4.3 showed that arrow laws (5)–(9) correspond precisely to this monoid having internal strength. Since both notions have been defined more generally than just for the Haskell category **HT**, we can now lift these properties into our main definition.

Definition 4.2
Let $\mathbf{C}$ be a small category with finite products. An *arrow* over $\mathbf{C}$ is a monoid in $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$ whose carrier has an internal strength.

In Appendix A we extend the definition of arrow to bifunctors $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$, where $\mathbf{C}$ is $\mathbf{V}$-enriched and both categories satisfy suitable size restrictions.

The combination of Propositions 4.2 and 4.3 justifies this categorical definition by showing that in the small category **HT** of Haskell types and functions our categorical notion of arrows coincides with the conventional one. Let us record this formally.

Corollary 4.1
An instantiation $(A, \ggg, arr, first)$ of the Haskell arrow class is an arrow over **HT** in the sense of Definition 4.2. $\square$

Summarising, we have shown that arrows in a type theoretic setting coincide with monoids in the category of bifunctors $\mathbf{HT}^{\mathrm{op}} \times \mathbf{HT} \to \mathbf{Set}$ with internal strength. We then lifted this property to a definition of arrow on any suitable category of bifunctors $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$, of which we described the case $\mathbf{V} = \mathbf{Set}$ explicitly.

5 Biarrows

The aim of this section is to illustrate that our categorical semantics for Haskell arrows is at the right level of abstraction. We consider the example of so-called biarrows, the semantics of which can now be elegantly expressed by simply restricting the underlying category.

Biarrows were introduced in (Alimarine *et al.*, 2005) as a language extension facilitating bidirectional computations. For example, implementing a parser as a biarrow begets a pretty-printer "for free". In Haskell terms, a biarrow is a further restriction of the arrow class interface:

$$\begin{array}{llcl} class & Arrow\ B & => & BiArrow\ B\ where \\ & (\leftrightarrow) & :: & (X \to Y) \to (Y \to X) \to B\ X\ Y \\ & inv & :: & B\ X\ Y \to B\ Y\ X \end{array}$$

satisfying (2), (5), (7), (9), and

$$(f_1 \leftrightarrow g_2) \ggg (g_1 \leftrightarrow f_2) = (f_1 \ggg g_1) \leftrightarrow (f_2 \ggg g_2), \tag{3'}$$
$$(\mathrm{id} \leftrightarrow \mathrm{id}) \ggg f = f = f \ggg (\mathrm{id} \leftrightarrow \mathrm{id}) \tag{4'}$$
$$first(h) \ggg (\mathrm{id} \times f) \leftrightarrow (\mathrm{id} \times g) = (\mathrm{id} \times f) \leftrightarrow (\mathrm{id} \times g) \ggg first(h) \tag{6'}$$
$$first(f \leftrightarrow g) = (f \times \mathrm{id}) \leftrightarrow (g \times \mathrm{id}) \tag{8'}$$
$$inv(inv(f)) = f \tag{22}$$
$$inv(f \ggg g) = inv(g) \ggg inv(f) \tag{23}$$
$$inv(f \leftrightarrow g) = g \leftrightarrow f \tag{24}$$
$$inv(first(f)) = first(inv(f)) \tag{25}$$

We see that biarrows require a further operation inv on top of arr, $\ggg$ and $first$, whose type should be $inv : B(X,Y) \to B(Y,X)$. Since we defined an arrow as a bifunctor of the form $B : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$, a natural transformation $B(X,Y) \to B(Y,X)$ is a problem because of the co- versus contravariance. The following definition enforces the required symmetry $\mathbf{C} = \mathbf{C}^{\mathrm{op}}$.

Definition 5.1
For $\mathbf{C}$ a category, define a full subcategory $\mathbf{C}_{\leftrightarrows}$ of $\mathbf{C}^{\mathrm{op}} \times \mathbf{C}$ by the class of objects $\{(X,X) : X \in \mathbf{C}\}$. We identify the objects of $\mathbf{C}_{\leftrightarrows}$ and $\mathbf{C}$, so that a morphism $X \to Y$ in $\mathbf{C}_{\leftrightarrows}$ is a pairs of morphisms $X \to Y$ and $Y \to X$ of $\mathbf{C}$.

The category $\mathbf{C}_{\leftrightarrows}$ is self-dual by construction. It has finite products if and only if $\mathbf{C}$ has finite biproducts – which, for this situation, are finite products and coproducts that coincide.

Definition 5.2
A *biarrow* on $\mathbf{C}$ is defined to be an arrow B on $\mathbf{C}_{\leftrightarrows}$ that is equipped with a natural transformation $inv : B \Rightarrow B^*$ where $B^* : \mathbf{C}_{\leftrightarrows}^{\mathrm{op}} \times \mathbf{C}_{\leftrightarrows} \to \mathbf{Set}$ is given by $B^*(X,Y) = B(Y,X)$ and $\langle (g,f),(h,k) \rangle \mapsto \langle (f,g),(k,h) \rangle$. The components $inv_{X,Y} : B(X,Y) \to B(Y,X)$ are required to satisfy:

$$inv \circ inv = \mathrm{id} \tag{26}$$
$$inv \circ \ggg \circ \gamma = \ggg \circ (inv \times inv) \tag{27}$$
$$inv \circ arr = arr \circ \gamma \tag{28}$$
$$inv \circ first = first \circ inv \tag{29}$$

A biarrow B is called *left-invertible* on $b \in B(X,Y)$ if $b \ggg inv(b) = \mathrm{id}_X$. It is called *right-invertible* on b if $inv(b) \ggg b = \mathrm{id}_Y$, and *invertible* if it is both left- and right-invertible.

Notice that the domain of the operation $arr : \mathbf{C}_{\leftrightarrows}(X,Y) \to B(X,Y)$ of a biarrow B is $\mathbf{C}(X,Y) \times \mathbf{C}(Y,X)$. Hence one can model the operation $(\leftrightarrow)$ simply by the unit arr of the monoidal structure of a biarrow.

Let us conclude this section by studying under which conditions the examples of Section 2.3 are (invertible) biarrows.

Proposition 5.1
The arrow $\mathbf{C}^{\mathrm{op}}_{\leftrightarrows} \times \mathbf{C}_{\leftrightarrows} \to \mathbf{Set}$ of pure functions, given by $B(X,Y) = \mathbf{C}(X,Y) \times \mathbf{C}(Y,X)$, is a biarrow. On morphisms it is given by:

$$\Big(X' \underset{g}{\overset{f}{\leftrightarrows}} X,\ Y \underset{k}{\overset{h}{\rightleftarrows}} Y'\Big) \longmapsto \lambda(a,b) \in B(X,Y)\,.\,(h \circ a \circ g, f \circ b \circ k)$$

It is left-invertible on (the subcategories of $\mathbf{C}_{\leftrightarrows}$ of) split monics, right-invertible on split epis, and invertible on isomorphisms.

Proof
Use the "swap" map $(a,b) \mapsto (b,a)$ as the natural transformation *inv*. Since this is in fact a natural isomorphism, (26)–(29) are trivially satisfied. Left-invertibility on a morphism $b \in B(X,Y)$ means precisely that its components $b_1 : X \to Y$ and $b_2 : Y \to X$ satisfy $b_2 \circ b_1 = \mathrm{id}_X$, proving the claim that b_1 be split mono. The claim on split epis is similar, and a morphism that is split mono as well as split epi is necessarily an isomorphism. □

In the style of Section 3, one may proceed to formulate a calculus of biarrows. In this way one may structurally develop programs together with their "inverses", as in (Alimarine *et al.*, 2005).

6 Kleisli and Eilenberg-Moore constructions for arrows

In Section 4 we have observed that arrows are monoids in a category of bifunctors $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$, where $\mathbf{C}$ is $\mathbf{V}$-enriched. This perspective bears a resemblance to monads. After all, monads are also monoids in a functor category, namely $\mathbf{Cat}(\mathbf{C},\mathbf{C})$. Pursuing this analogy, this section investigates Eilenberg-Moore and Kleisli constructions for arrows.

We should warn the reader that this section requires a stronger stomach for technical/categorical details. For clarity of exposition, we focus on a non-enriched setting throughout this section, *i.e.* $\mathbf{V} = \mathbf{Set}$. Enriching the whole framework in a monoidal closed category $\mathbf{V}$ (following Appendix A) is then straightforward. Concretely, this means that the results in this section will be needlessly restricted to locally small categories.

6.1 Arrows are Freyd categories

Let us start by exhibiting an obvious way to associate a category of "(structured) computations" with an arrow. This construction will subsequently be shown to

- give Freyd categories (Robinson & Power, 1997; Power & Thielecke, 1997);
- provide a bijective correspondence between arrows and Freyd categories, adhering to the slogan "arrows are Freyd categories" (Heunen & Jacobs, 2006);
- be the Kleisli construction for arrows, in a suitable 2-categorical sense, under the motto "Freyd is Kleisli, for arrows" (Jacobs & Hasuo, 2006).

Definition 6.1 (*The category* $\mathbf{C}_A$)

Let $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ be an arrow, with operations $arr, \ggg$ and $first$. Define a category $\mathbf{C}_A$ to have objects those of $\mathbf{C}$, and morphisms $X \to Y$ given by elements of the set $A(X, Y)$. Identities and composition are given by arr and $\ggg$ in the obvious manner.

Before proceeding to explain why this construction gives Freyd categories in Theorem 6.1 below, let us briefly summarize what a Freyd category is. For that, we need the notion of a premonoidal category, which can intuitively be thought of as a monoidal category in which the tensor need not be a bifunctor, though it is functorial in each variable separately.

Definition 6.2
A *binoidal category* is a category $\mathbf{D}$, with for every object X two functors $(-) \ltimes X : \mathbf{D} \to \mathbf{D}$ and $X \rtimes (-) : \mathbf{D} \to \mathbf{D}$ such that $X \ltimes Y = X \rtimes Y$. Hence we write $X \boxtimes Y = X \ltimes Y = X \rtimes Y$. A morphism f is called *central* if for each g, both

- $(f \ltimes \mathrm{id}) \circ (\mathrm{id} \rtimes g) = (\mathrm{id} \rtimes g) \circ (f \ltimes \mathrm{id})$, and
- $(\mathrm{id} \rtimes f) \circ (g \ltimes \mathrm{id}) = (g \ltimes \mathrm{id}) \circ (\mathrm{id} \rtimes f)$.

For such a central f it makes sense to write $f \boxtimes g$ or $g \boxtimes f$ for these composites.

Definition 6.3
A *symmetric premonoidal category* is a binoidal category $\mathbf{D}$ together with an object $I \in \mathbf{D}$ and natural isomorphisms with central components $\alpha : (X \boxtimes Y) \boxtimes Z \to X \boxtimes (Y \boxtimes Z)$, $\lambda : I \boxtimes X \to X$, $\rho : X \boxtimes I \to X$ and $\gamma : X \boxtimes Y \to Y \boxtimes X$ that obey the familiar coherence properties for monoidal categories.

The non-bifunctoriality reflects the order of side-effects when we think of $\mathbf{D}$ as a category of 'computations'. When we include a category $\mathbf{C}$ of 'values', we arrive at the notion of a Freyd category (Robinson & Power, 1997; Power & Thielecke, 1997; Levy *et al.*, 2003).

Definition 6.4
A *Freyd category* consists of a symmetric premonoidal category $\mathbf{D}$ together with a category $\mathbf{C}$ with finite products, and an identity-on-objects functor $J : \mathbf{C} \to \mathbf{D}$ which:[2]

- carries Cartesian products in $\mathbf{C}$ to premonoidal products in $\mathbf{D}$ on-the-nose: $J(X \times Y) = X \boxtimes Y$, $J(\alpha) = \alpha$, $J(\lambda) = \lambda$, etc.;
- preserves central maps. Since every morphism in $\mathbf{C}$ is central, this just means that every morphism in $\mathbf{D}$ of the form $J(g)$ is central.

A Freyd category $\mathbf{C} \to \mathbf{D}$ is called (locally) small if the category $\mathbf{D}$ is (locally) small.

[2] Such a functor J satisfying the two conditions is called a *strict premonoidal functor* in Definition 8, (Power & Thielecke, 1997). In that paper the notion of Freyd category has not yet been given its name; still it is a central notion there and appears e.g. in Theorem 14.

Returning to the central construction of this section, the category $\mathbf{C}_A$ of Definition 6.1, observe that there is an identity-on-objects functor $J_A : \mathbf{C} \to \mathbf{C}_A$ whose action on morphisms is given by *arr*. This forms an instance of a Freyd category, as Theorem 6.1 will prove. We shall call this mapping $A \mapsto (\mathbf{C} \overset{J_A}{\to} \mathbf{C}_A)$ from arrows to Freyd categories the *Kleisli construction* for arrows. Although this name will be fully justified (2-categorically) in Section 6.3, we can observe now already some similarities to the Kleisli construction for monads. For an arrow A induced by a (co)monad, the associated Freyd category $\mathbf{C}_A$ coincides with the (co)Kleisli category for the (co)monad in the usual sense. For the arrow $\overline{(N, M)}$ of (12), induced by both a monad M and a comonad N, the Kleisli construction yields what is called the bi-Kleisli category used *e.g.* in (Uustalu & Vene, 2005).

The Kleisli construction $A \mapsto (\mathbf{C} \overset{J_A}{\to} \mathbf{C}_A)$ turns out to be a bijective map from arrows to Freyd categories. This observation turns the oft-heard (informal) statement "arrows are Freyd categories", into the following concrete theorem.

Theorem 6.1 ("Arrows are Freyd categories")
For a locally small category $\mathbf{C}$ with finite products, there is a one-to-one correspondence between arrows A over $\mathbf{C}$ (in the sense of Definition 4.2) and locally small Freyd categories $\mathbf{C} \to \mathbf{D}$.

Proof
Suppose we are given an arrow $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ with operations $\ggg$, *arr*, and ist. Putting $\mathbf{D} = \mathbf{C}_A$ entails that $\mathbf{D}$ is symmetric premonoidal by $I = 1 \in \mathbf{D}$ and $X \boxtimes Y = X \times Y$. The premonoidal tensor $\boxtimes$ extends to a functor (on morphisms) by virtue of the provided ist (or equivalently $first$, see Proposition 4.3, and *second*), since every morphism $a \in A(X, Y)$ yields $a \boxtimes Z = first_Z(a) : X \boxtimes Z \to Y \boxtimes Z$ and $Z \boxtimes a = second_Z(a) : Z \boxtimes X \to Z \boxtimes Y$. The transformation of an arrow into a Freyd category is completed by defining $J : \mathbf{C} \to \mathbf{D}$ to act as the identity on objects, and as *arr* on morphisms.

Conversely, suppose given a Freyd category $J : \mathbf{C} \to \mathbf{D}$. Define $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ by $A(X, Y) = \mathbf{D}(X, Y)$. This A is made into a monoid in $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$ by the unit $arr = J : \mathbf{C}(X, Y) \to \mathbf{D}(X, Y)$, and for multiplication $\ggg$ the composition $A(X, P) \times A(P, Y) \to A(X, Y)$ in $\mathbf{D}$. Furthermore we can define internal strength

$$\mathrm{ist}_{X,Y} : A(X, Y) \to A(X, Y \times X) \quad \text{by} \quad \mathrm{ist}_{X,Y}(f) = (f \ltimes X) \circ J(\langle \mathrm{id}, \mathrm{id} \rangle).$$

Naturality of ist in Y is obvious, dinaturality in X boils down to the fact that the diagram

$$\begin{tikzcd}
& \mathbf{D}(X,Y) \arrow[r, "\mathrm{ist}_{X,Y}"] & \mathbf{D}(X, Y \boxtimes X) \arrow[dr, "(Y \boxtimes J(g)) \circ (-)"] & \\
\mathbf{D}(X',Y) \arrow[ur, "(-) \circ J(g)"] \arrow[dr, equal] & & & \mathbf{D}(X, Y \boxtimes X') \\
& \mathbf{D}(X',Y) \arrow[r, "\mathrm{ist}_{X',Y}"'] & \mathbf{D}(X', Y \boxtimes X') \arrow[ur, "(-) \circ J(g)"'] &
\end{tikzcd}$$

commutes for every morphism $g : X \to X'$ in $\mathbf{C}$. The crux here is that it need only commute for morphisms g of $\mathbf{C}$, *i.e.* morphisms of $\mathbf{D}$ of the form $J(g)$ (cf. equations (20) and (21)), which are central. Since one also readily checks (16)–(19), this proves that a Freyd category induces an arrow. $\square$

The proof of the previous theorem reminds one strongly of the situation for monads. In the well-known correspondence between

- monoids in the category of functors $\mathbf{C} \to \mathbf{C}$,
- monads M on $\mathbf{C}$, and
- identity-on-objects functors $J : \mathbf{C} \to \mathbf{D}$ that have a right adjoint,

the functor J arises from M by the Kleisli construction, while J induces a monad M by the adjunction. The proof of the above theorem is a generalisation of this correspondence (Heunen & Jacobs, 2006).

6.2 Eilenberg-Moore algebras for arrows

After considering Kleisli constructions for arrows in the previous subsection, we now turn to the notion of Eilenberg-Moore algebras for arrows. We aim for two properties of this new notion. First, for arrows induced by (co)monads it should coincide with the usual notion of (co)algebra. Secondly, an arrow-algebra should be a retraction of a Kleisli category, much like for monads.

Let us start by the situation for monads. We shall understand (Eilenberg-Moore) (co)algebras in 2-categorical style as natural transformations. Explicitly, an algebra for a monad (T, η, μ) on a category $\mathbf{C}$ is a map $\varphi : T \Rightarrow \mathrm{id}_{\mathbf{C}}$ satisfying the familiar equations $\varphi \circ \eta = \mathrm{id}$ and $\varphi \circ T\varphi = \varphi \circ \mu$. Since such a monad T is the same thing as a monoid in $\mathbf{Cat}(\mathbf{C}, \mathbf{C})$, with monoidal structure given by functor composition and the identity functor, these monad equations boil down to the following commuting diagram.

$$\begin{tikzcd}
\mathrm{id}_{\mathbf{C}} \arrow[r, "\eta"] \arrow[dr, equal] & T \arrow[d, "\varphi"'] & T \otimes T = T^2 \arrow[l, "\mu"'] \arrow[d, "\varphi\otimes\varphi=\varphi\circ T\varphi"] \\
 & \mathrm{id}_{\mathbf{C}} \arrow[r, equal] & \mathrm{id}_{\mathbf{C}} \otimes \mathrm{id}_{\mathbf{C}}
\end{tikzcd}$$

When T is strong we might as well require coherence with its strength.

$$\begin{tikzcd}
T(\cdot) \times (*) \arrow[r, "\varphi"] \arrow[dr, "\varphi\times\mathrm{id}"'] & T((\cdot) \times (*)) \arrow[d, "\varphi"] \\
 & (\cdot) \times (*)
\end{tikzcd} \tag{30}$$

The fact that an arrow is also a monoid – in a category of bifunctors – leads to the following definition.

Definition 6.5
Let $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$ be an arrow. An *algebra* for A is a natural transformation $\chi : A \Rightarrow \mathrm{Hom}$ that is compatible with arr, $\ggg$ and $\langle first \rangle$, in the sense that the

following diagrams commute for each $Z \in \mathbf{C}$.

$$
\begin{tikzcd}
\mathrm{Hom} \arrow[r, "arr"] \arrow[dr, equal] & A \arrow[d, "\chi"] & A \otimes A \arrow[l, "\ggg"'] \arrow[d, "\chi\otimes\chi"] \\
& \mathrm{Hom} & \mathrm{Hom} \otimes \mathrm{Hom} \arrow[l, "\cong"]
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "\langle first \rangle"] \arrow[d, "\chi"'] & A_{\times} \arrow[d, "\chi"] \\
\mathrm{Hom} \arrow[r, "\langle (-)\times Z \rangle_Z"'] & \mathrm{Hom}_{\times},
\end{tikzcd}
\tag{31}
$$

where we used the functor $A_\times$ from Lemma 4.2 and the analogous $\mathrm{Hom}_\times(X, Y) = \mathrm{Hom}(X \times Z, Y \times Z)$.

An algebra of an arrow is thus a (natural) mapping of computations to (pure) functions. One might have expected a single mapping $\chi : A(X, Y) \to \mathrm{Hom}(X, Y)$, but then it is unclear how to capture commutation with composition $\ggg$. Specifically, defining $\chi \otimes \chi$ in (31) is problematic.

In elementary terms, an algebra χ must thus satisfy the equations

$$\chi(arr(f)) = f, \quad \chi(a \ggg b) = \chi(b) \circ \chi(a), \quad \chi(first(a)) = \chi(a) \times \mathrm{id}.$$

Naturality then is a consequence:

$$
\begin{aligned}
\chi(A(f, g)(a)) &= \chi(arr(f) \ggg a \ggg arr(g)) \\
&= \chi(arr(g)) \circ \chi(a) \circ \chi(arr(f)) \\
&= g \circ \chi(a) \circ f \\
&= \mathrm{Hom}(f, g)(\chi(a)).
\end{aligned}
$$

The rest of this subsection is devoted to results stemming from Definition 6.5. The first one concerns arrows $\overline{M}$ induced by monads M as in (10). This subsection contains two similar results. We write out the proof of this first one in full detail, to distinguish the trivial parts from the non-trivial ones. Especially the order in which to apply the equations in the verification of the multiplication law in the proof below is delicate.

Proposition 6.1

For a strong monad M there is a bijective correspondence between strong-monad-algebras $M \Rightarrow \mathrm{id}$ and arrow-algebras $\overline{M} \Rightarrow \mathrm{Hom}$.

Proof

Given a monad algebra $\varphi : M \Rightarrow \mathrm{id}$, define $\overline{\varphi} : \overline{M} \Rightarrow \mathrm{Hom}$ on $a \in \mathrm{Hom}(X, MY)$ by $\overline{\varphi}_{(X,Y)}(a) = \varphi_Y \circ a$. We check that it satisfies the required properties, omitting subscripts for clarity.

$$\begin{aligned}
\overline{\varphi}(arr(f)) &= \varphi \circ \eta \circ f \\
&= f, \\
\overline{\varphi}(a \ggg b) &= \varphi \circ \mu \circ M(b) \circ a \\
&= \varphi \circ M(\varphi \circ b) \circ a \\
&= (\varphi \circ b) \circ (\varphi \circ a) \\
&= \overline{\varphi}(b) \circ \overline{\varphi}(a),
\end{aligned}
\qquad
\begin{aligned}
\overline{\varphi}(first_Z(a)) &= \varphi \circ \mathrm{st} \circ (a \times Z) \\
&= (\varphi \times Z) \circ (a \times Z) \\
&= (\varphi \circ a) \times Z) \\
&= \overline{\varphi}(a) \times Z, \\
\overline{\varphi}(\overline{M}(f,g)(a)) &= \overline{\varphi}(M(g) \circ a \circ f) \\
&= \varphi \circ M(g) \circ a \circ f \\
&= g \circ \varphi \circ a \circ f \\
&= \mathrm{Hom}(f,g)(\varphi \circ a) \\
&= \mathrm{Hom}(f,g)(\overline{\varphi}(a)).
\end{aligned}$$

Conversely, given an arrow-algebra $\chi : \overline{M} \Rightarrow \mathrm{Hom}$ we define $\overline{\chi}_X : MX \to X$ as $\chi_{(MX,X)}(\mathrm{id}_{MX})$. This definition suggests that the Yoneda Lemma can also be used. For clarity, we have chosen to write out the proof that we get an M-algebra directly. The unit-law and naturality are easy:

$$\begin{aligned}
\overline{\chi} \circ \eta &= \chi(\mathrm{id}) \circ \chi(arr(\eta)) \\
&= \chi(arr(\eta) \ggg \mathrm{id}) \\
&= \chi(\mu \circ M(\mathrm{id}) \circ \eta \circ \eta) \\
&= \chi(\eta) \\
&= \chi(arr(\mathrm{id})) \\
&= \mathrm{id},
\end{aligned}
\qquad
\begin{aligned}
\overline{\chi} \circ M(f) &= \chi(\mathrm{id}) \circ \chi(arr(M(f))) \\
&= \chi(arr(M(f)) \ggg \mathrm{id}) \\
&= \chi(\mu \circ M(\mathrm{id}) \circ \eta \circ M(f)) \\
&= \chi(M(f)) \\
&= \chi(\mu \circ M(\eta \circ f) \circ \mathrm{id}) \\
&= \chi(\mathrm{id} \ggg arr(f)) \\
&= \chi(arr(f)) \circ \chi(\mathrm{id}) \\
&= f \circ \overline{\chi}.
\end{aligned}$$

Compatibility (30) with strength is proved as follows.

$$\begin{aligned}
\overline{\chi} \times Y = \chi(\mathrm{id}) \times Y &\overset{(*)}{=} \chi(\mathrm{st} \circ (\mathrm{id} \times \mathrm{id})) = \chi(\mathrm{st} \circ \mathrm{id}) \\
= \chi(\mathrm{id} \circ \mathrm{st}) &\overset{(**)}{=} \chi(\mathrm{id}) \circ \mathrm{st} = \overline{\chi} \circ \mathrm{st}.
\end{aligned}$$

The equality $(*)$ holds because χ is compatible with $first$, given by $first(a) = \mathrm{st} \circ (a \times \mathrm{id})$; $(**)$ uses the naturality of χ.

Verification of the multiplication law is more subtle.

$$\begin{aligned}
\overline{\chi} \circ M(\overline{\chi}) &= \chi(\mathrm{id}) \circ \chi(arr(M(\overline{\chi}))) = \chi(arr(M(\overline{\chi})) \ggg \mathrm{id}) \\
&= \chi(\mu \circ M(\mathrm{id}) \circ \eta \circ M(\overline{\chi})) = \chi(M(\overline{\chi})) = \chi(\mu \circ M(\eta \circ \overline{\chi}) \circ \mathrm{id}) \\
&= \chi(arr(\overline{\chi}) \ggg \mathrm{id}) = \chi(\mathrm{id}) \circ \chi(arr(\overline{\chi})) = \chi(\mathrm{id}) \circ \chi(\mathrm{id}) \\
&= \chi(\mathrm{id} \ggg \mathrm{id}) = \chi(\mu \circ M(\mathrm{id}) \circ \mathrm{id}) = \chi(\mu) \\
&= \chi(\mu \circ M(\mathrm{id}) \circ \eta \circ \mu) = \chi(arr(\mu) \ggg \mathrm{id}) \\
&= \chi(\mathrm{id}) \circ \chi(arr(\mu)) = \overline{\chi} \circ \mu.
\end{aligned}$$

The proof is completed by checking that the correspondence is indeed bijective:

$$\begin{aligned} \overline{\overline{\varphi}} &= \overline{\varphi}(\mathrm{id}) \\ &= \varphi \circ \mathrm{id} \\ &= \varphi \end{aligned} \qquad\qquad \begin{aligned} \overline{\overline{\chi}}(a) &= \overline{\chi} \circ a \\ &= \chi(\mathrm{id}) \circ \chi(arr(a)) \\ &= \chi(arr(a) \ggg \mathrm{id}) \\ &= \chi(\mu \circ M(\mathrm{id}) \circ \eta \circ a) \\ &= \chi(a). \quad \square \end{aligned}$$

There is a dual result for comonads. It shows that arrow-algebras form a common generalisation of monad algebras and comonad coalgebras. The proof is similar to the one above, and left to the reader.

Proposition 6.2
For a comonad N there is a bijective correspondence between comonad-coalgebras $\mathrm{id} \Rightarrow N$ and arrow-algebras $\overline{N} \Rightarrow \mathrm{Hom}$. $\square$

The previous two propositions can be extended to bi-algebras.

Proposition 6.3
Let M be a strong monad, N a comonad, and $\lambda : NM \Rightarrow MN$ a distributive law between them. Then there is a bijective correspondence between λ-bi-algebras $M \Rightarrow \mathrm{id} \Rightarrow N$ and arrow-algebras $\overline{(M, N)} \Rightarrow \mathrm{Hom}$ as in (12).

Proof
We shall only give the essentials, and leave details to the reader. Assuming a λ-bialgebra $M \overset{\varphi}{\Longrightarrow} \mathrm{id} \overset{\psi}{\Longrightarrow} N$, the following diagram commutes by definition.

$$\begin{tikzcd} M \arrow[r, "\varphi"] \arrow[d, "\psi M"'] & \mathrm{id} \arrow[r, "\psi"] & N \\ NM \arrow[rr, "\lambda"'] & & MN \arrow[u, "\varphi N"'] \end{tikzcd}$$

We obtain an arrow-algebra $\overline{(\varphi, \psi)} : \overline{(M, N)} \Rightarrow \mathrm{Hom}$ by $\overline{(\varphi, \psi)}(a) = \varphi \circ a \circ \psi$. It satisfies the required equations.

Conversely, an arrow-algebra $\chi : \overline{(M, N)} \Rightarrow \mathrm{Hom}$ induces a pair of maps $\overline{\chi} = (\overline{\chi}_1, \overline{\chi}_2)$ by

$$\begin{array}{rcccl} \overline{\chi}_1 & = & \chi_{(MX,X)}(\varepsilon_{MX}) & : & MX \longrightarrow X \\ \overline{\chi}_2 & = & \chi_{(X,NX)}(\eta_{NX}) & : & X \longrightarrow NX. \end{array}$$

Of the verification that it yields an appropriate λ-bialgebra, we only show commutation of the above diagram, *i.e.* that we have a compatible algebra-coalgebra pair.

$$\begin{aligned} \overline{\chi}_1 \circ \lambda \circ \overline{\chi}_2 &= \chi(\varepsilon) \circ \chi(arr(\lambda)) \circ \chi(\eta) = \chi(\eta \ggg arr(\lambda) \ggg \varepsilon) \\ &= \chi([\mu \circ M(\eta \circ \lambda \circ \varepsilon) \circ \lambda \circ N(\eta) \circ \delta] \ggg \varepsilon) \\ &= \chi([\eta \circ \lambda] \ggg \varepsilon) = \chi(\mu \circ M(\varepsilon) \circ \lambda \circ N(\eta \circ \lambda) \circ \delta) \\ &= \chi(\lambda) = \chi(\mu \circ M(\eta) \circ \lambda \circ N(\varepsilon) \circ \delta) \\ &= \chi(\varepsilon \ggg \eta) = \chi(\eta) \circ \chi(\varepsilon) = \overline{\chi}_2 \circ \overline{\chi}_1. \quad \square \end{aligned}$$

The next Lemma shows that arrow algebras are retractions of Kleisli categories.

Lemma 6.1
An algebra for an arrow A on $\mathbf{C}$ is precisely a left inverse $K : \mathbf{C}_A \to \mathbf{C}$ of the "Kleisli inclusion" functor $J_A : \mathbf{C} \to \mathbf{C}_A$. (A left inverse is sometimes also called a retraction, and means $KJ = \mathrm{id}$.)

Proof
Given an algebra $\chi : A \Rightarrow \mathrm{Hom}$ we get a functor $\mathbf{C}_A \to \mathbf{C}$ by $X \mapsto X$ and $a \mapsto \chi(a)$. It forms a retraction because $\chi(arr(f)) = f$. Conversely, a retraction $K : \mathbf{C}_A \to \mathbf{C}$ yields an algebra $\overline{K} : A \Rightarrow \mathrm{Hom}$ by $a \mapsto K(a)$. We check naturality:

$$\begin{aligned}\overline{K}(A(f,g)(a)) &= K(arr(f) \ggg a \ggg arr(g))\\ &= K(arr(g)) \circ K(a) \circ K(arr(f))\\ &= KJ(g) \circ K(a) \circ KJ(f)\\ &= g \circ K(a) \circ f\\ &= \mathrm{Hom}(f,g)(\overline{K}(a)). \quad \square\end{aligned}$$

The previous Lemma justifies the term Eilenberg-Moore algebra for arrows, since the next Lemma gives an analogous characterisation for monads.

Lemma 6.2
For a monad M on a category $\mathbf{C}$, there is a bijective correspondence between algebras $M \Rightarrow \mathrm{id}$ and retractions of the Kleisli inclusion $J : \mathbf{C} \to \mathbf{C}_M$.

Proof
Given an algebra $\varphi : M \Rightarrow \mathrm{id}$, define a functor $\overline{\varphi} : \mathbf{C}_M \to \mathbf{C}$ by $X \mapsto X$ and $f \mapsto \varphi \circ f$. This clearly yields a functor, and a moreover a retraction:

$$\overline{\varphi}(J(f)) = \overline{\varphi}(\eta \circ f) = \varphi \circ \eta \circ f = f.$$

Conversely, given a retraction $K : \mathbf{C}_M \to \mathbf{C}$ of J, we define $\overline{K} : M \Rightarrow \mathrm{id}$ via $\overline{K}_X = K(\mathrm{id}_{MX} : MX \to X) : MX \to X$. This yields a natural transformation, and a monad algebra:

$$\begin{aligned}\overline{K}_X \circ \eta_X &= K(\mathrm{id}) \circ KJ(\eta)\\ &= K(\mu \circ M(\mathrm{id}) \circ \eta \circ \eta)\\ &= K(\eta)\\ &= \mathrm{id},\end{aligned} \qquad \begin{aligned}\overline{K}_X \circ M(\overline{K}_X) &= \overline{K}_X \circ \overline{K}_{MX}\\ &= K(\mathrm{id}) \circ K(\mathrm{id})\\ &= K(\mu \circ M(\mathrm{id}) \circ \mathrm{id})\\ &= K(\mu)\\ &= K(\mu \circ M(\mathrm{id}) \circ \eta \circ \mu)\\ &= K(\mathrm{id}) \circ KJ(\mu)\\ &= \overline{K}_X \circ \mu. \quad \square\end{aligned}$$

6.3 Freyd is Kleisli, for arrows

For monads and comonads, the Kleisli construction is characterised 2-categorically as a certain left 2-adjoint (Street, 1972). Theorem 6.2 will prove that the (bijective) mapping $A \mapsto (\mathbf{C} \to \mathbf{C}_A)$ that we have been looking at allows a similar

2-categorical characterisation. Therefore the bijective mapping is justifiably called the Kleisli construction for arrows. This subsection will extend the notion of arrow on a category with finite products to arrows on Freyd categories in general, and studies some additional (2-categorical) properties. It assumes a reasonable level of familiarity with 2-categories; we refer to (Borceux, 1994a) for details.

Let us first recall the situation for monads. The Kleisli construction is the left 2-adjoint of the canonical "insertion" 2-functor *Ins* in the following 2-adjunction.[3]

$$\begin{tikzcd} \mathbf{Cat} \arrow[r, bend left, "Ins"] \arrow[r, phantom, "\top"] & \mathbf{Mnd}(\mathbf{Cat}^*)^* \arrow[l, bend left, "\mathcal{K}\ell"] \end{tikzcd} \tag{32}$$

Here the 2-category $\mathbf{Mnd}(\mathbf{Cat}^*)^*$ is such that:

- An object is a pair $(\mathbf{C}, M)$ of a category $\mathbf{C}$ and a monad M on it;
- A 1-cell $(H, \sigma) : (\mathbf{C}, M) \to (\mathbf{D}, M')$ is a pair of a functor $H : \mathbf{C} \to \mathbf{D}$ and a natural transformation $\sigma : HM \Rightarrow M'H$, which is compatible with monad structures of M and M'.

$$\begin{tikzcd} \mathbf{C} \arrow[r, "H"] \arrow[d, "M"'] \arrow[dr, phantom, "\sigma \Nearrow"] & \mathbf{D} \arrow[d, "M'"] \\ \mathbf{C} \arrow[r, "H"'] & \mathbf{D} \end{tikzcd}$$

- A 2-cell $\alpha : (H, \sigma) \Rightarrow (H', \sigma')$, is a natural transformation $\alpha : H \Rightarrow H'$ which is compatible with σ and σ' in a suitable sense.

The functor *Ins* is a canonical one mapping an object $\mathbf{C}$ to $(\mathbf{C}, \mathrm{id})$. The functor $\mathcal{K}\ell$ of the Kleisli construction maps an object $(\mathbf{C}, T)$ to the Kleisli category $\mathbf{C}_T$.

Let us go through similar 2-categorical motions for arrows. Denote by **FPCat** the 2-category of categories with finite products. At first, one may try for a functor $\mathcal{K}\ell : \mathbf{Arr}_{\mathbf{FPCat}} \to \mathbf{FPCat}$ for a suitably defined 2-category $\mathbf{Arr}_{\mathbf{FPCat}}$ of arrows,[4] which maps $(\mathbf{C}, A)$ to $\mathbf{C}_A$. However, the category $\mathbf{C}_A$ does not necessarily have finite products; it has only the weaker structure of a Freyd category. The same difficulty occurs already attempting to extend the 2-adjunction (32) to strong monads. The problem is resolved by considering arrows on Freyd categories. The 2-adjunction

[3] The notation $\mathbf{Mnd}(\mathbf{Cat}^*)^*$ originates in (Street, 1972); see its Section 4. The operator $(\_)^*$ on 2-categories opposes 1-cells (but not 2-cells). The constructor **Mnd** is actually a 2-functor which maps a 2-category $\mathbf{C}$ to the "2-category of monads" on $\mathbf{C}$.

[4] We use the notation $\mathbf{Arr}_{\mathbf{FPCat}}$ rather than $\mathbf{Arr}(\mathbf{FPCat})$. The notion of monad is defined in any 2-category $\mathbf{C}$ hence the notation $\mathbf{Mnd}(\mathbf{C})$ makes sense; whereas the notion of arrow does not come with such generality.
Moreover, two $*$'s in $\mathbf{Mnd}(\mathbf{Cat}^*)^*$ are gone in the corresponding $\mathbf{Arr}_{\mathbf{FPCat}}$. The two $*$'s were there due to the choice of "lax" monad morphisms as 1-cells in $\mathbf{Mnd}(\mathbf{C})$ (which is convenient for the Eilenberg-Moore construction); to have "oplax" monad morphisms instead as 1-cells we needed two $*$'s. In defining the "category of arrows" $\mathbf{Arr}_{\mathbf{FPCat}}$ there is no room for such choice between lax and oplax. For example, in the diagram (C 1) the 2-cell $\mathrm{Hom}(H)$ *must* be in this direction and not the other.

(32) for arrows then looks as follows.

$$\begin{tikzcd} \mathbf{Freyd} \arrow[r, bend left, "Ins"] \arrow[r, phantom, "\top"] & \mathbf{Arr}_{\mathbf{Freyd}} \arrow[l, bend left, "\mathcal{K}\ell"] \end{tikzcd} \tag{33}$$

The definition of arrows on Freyd categories is exactly the same as on categories with finite products, except for the conditions on $first$. Recall that in a Freyd category $\mathbf{C} \xrightarrow{J} \mathbf{K}$, the category $\mathbf{K}$ has a premonoidal structure denoted by $\boxtimes$.

Definition 6.6 (*Arrows on Freyd categories*)
An *arrow* on a Freyd category $\mathbf{C} \xrightarrow{J} \mathbf{K}$ is a monoid $A \otimes A \overset{\ggg}{\to} A \overset{arr}{\leftarrow} \mathrm{Hom}$ in the monoidal category $[\mathbf{K}^{\mathrm{op}} \times \mathbf{K}, \mathbf{Set}]$ of bifunctors, equipped with morphisms

$$first_{X,Y,Z} : A(X,Y) \to A(X \boxtimes Z, Y \boxtimes Z) \ ,$$

that are natural in X, Y and dinatural in Z, and satisfy the following equations.

$$first(a) \ggg arr(J(\pi_1)) = arr(J(\pi_1)) \ggg a \tag{5'}$$

$$first(a) \ggg arr(\mathrm{id} \boxtimes J(f)) = arr\ (\mathrm{id} \boxtimes J(f)) \ggg first(a) \tag{6'}$$

$$first(first(a)) \ggg arr(\alpha) = arr(\alpha) \ggg first(a) \tag{7'}$$

$$first(arr(g)) = arr(g \boxtimes \mathrm{id}) \tag{8'}$$

$$first(a \ggg b) = first(a) \ggg first(b) \tag{9'}$$

The conditions (5')–(9') correspond to (5)–(9) in the original definition. The equations (2)–(4) are already incorporated by the requirement that A be a monoid. Because J preserves premonoidal structure, the associativity isomorphisms α in (7') are inherited from $\mathbf{C}$ as $\alpha^{\mathbf{K}} = J(\alpha^{\mathbf{C}})$. Recalling the intuition that a morphism in $\mathbf{C}$ is a pure function while one in $\mathbf{K}$ is an effectful one, (6') requires only pure functions to commute with $first(a)$.

For arrows on categories with finite products, Proposition 4.3 establishes the equivalence between the operations $first$ and ist. This is also the case for arrows on Freyd categories, but here we prefer $first$.

The ingredients of the adjunction (33) are defined straightforwardly, although they become lenghty when spelled out, and there are some hidden subtleties in the details. We merely sketch the definitions here, and refer to Appendix C for the details.

The 2-categories **Freyd** and $\mathbf{Arr}_{\mathbf{Freyd}}$ are those of Freyd categories and arrows on Freyd categories, respectively. The 2-functor *Ins* carries an object $\mathbf{C} \xrightarrow{J} \mathbf{K}$ to the canonical Hom-arrow $(\mathbf{C} \xrightarrow{J} \mathbf{K}, \mathrm{Hom}_{\mathbf{K}})$. The 2-functor $\mathcal{K}\ell$ in the converse direction is essentially the Kleisli construction for arrows in Definition 6.1. Namely, an object $(\mathbf{C} \xrightarrow{J} \mathbf{K}, A)$ is mapped to a Freyd category $\mathbf{C} \xrightarrow{J} \mathbf{K} \xrightarrow{J_A} \mathbf{K}_A$, where $\mathbf{K} \xrightarrow{J_A} \mathbf{K}_A$ is constructed like in Definition 6.1.

Now we are ready to prove the (informal) claim “Freyd is Kleisli, for arrows”.

Theorem 6.2 (*“Freyd is Kleisli”*)
There is a 2-adjunction $\mathcal{K}\ell \dashv Ins : \mathbf{Arr}_{\mathbf{Freyd}} \to \mathbf{Freyd}$ as in diagram (33).

Proof
Its unit is given by $(\mathbf{C} \xrightarrow{J} \mathbf{K},\, A) \xrightarrow{(\mathrm{id}, J_A, \iota)} (\mathbf{C} \xrightarrow{J} \mathbf{K} \xrightarrow{J_A} \mathbf{K}_A,\, \mathrm{Hom}_{\mathbf{K}_A})$, where ι is the canonical natural transformation with components $\mathrm{id} : A(X,Y) \to \mathbf{K}_A(X,Y)$. $\square$

Arrows on categories with finite products also form a 2-category $\mathbf{Arr}_{\mathbf{FPCat}}$, just like $\mathbf{Arr}_{\mathbf{Freyd}}$. The obvious horizontal insertion 2-functors produce the following situation.

$$\begin{tikzcd}
\mathbf{Arr}_{\mathbf{Freyd}} \arrow[d, bend right, "\mathcal{K}\ell"'] \arrow[d, phantom, "\dashv"] & \mathbf{Arr}_{\mathbf{FPCat}} \arrow[l, "Ins'"'] \\
\mathbf{Freyd} \arrow[u, bend right, "Ins"'] & \mathbf{FPCat} \arrow[l, "Ins"] \arrow[u, "Ins"']
\end{tikzcd} \tag{34}$$

We conclude this section by elaborating this diagram. The following theorem gives its relation to the correspondence result of Theorem 6.1.

Theorem 6.3
The bijective correspondence of Theorem 6.1 between Freyd categories and arrows on categories with finite products extends to an isomorphism between 2-categories **Freyd** and $\mathbf{Arr}_{\mathbf{FPCat}}$ in an obvious way.

$$\begin{tikzcd}
\mathbf{Freyd} \arrow[r, bend left, "Arr"] \arrow[r, phantom, "\cong"] & \mathbf{Arr}_{\mathbf{FPCat}} \arrow[l, bend left, "\mathcal{K}\ell"]
\end{tikzcd} \quad \square$$

An intuition on an object $(\mathbf{C} \xrightarrow{J} \mathbf{K}, A)$ of $\mathbf{Arr}_{\mathbf{Freyd}}$ is that it has two different levels of extra computational structures added to $\mathbf{C}$. One is described by the Freyd category $\mathbf{C} \xrightarrow{J} \mathbf{K}$, and on top of it we have the other one expressed as the arrow A. But in fact, the additional expressive power that comes from having two infrastructures is essentially redundant. This can be put in precise 2-categorical terms, for the details of which we refer to (Jacobs & Hasuo, 2006).

7 Conclusion

Arrows are powerful tools in functional programming. They provide compositional infrastructure, relieving the programmer of tedious bookkeeping, and in fact enable more general interfaces than monadic programming.

The present article considered categorical denotational semantics for arrows. The pivotal point is Definition 4.2, characterising an arrow as a monoid in the category of bifunctors $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{Set}$, that moreover has a structure called internal strength.

This monoidal structure of arrows has been illustrated by several important real-world examples. Two of them were discussed leading up to the categorical definition in a way that does not require the arrow laws to be checked by hand. We have shown that a third language extension (biarrows) can be formulated elegantly using the provided semantics, indicating that they provide the right perspective and level of abstraction.

Exploiting the similarity to monads then led to Kleisli and Eilenberg-Moore constructions for arrows. The definitions have been supported by results analogous to that of (co)algebras for a (co)monad.

In fact, we have proven rigorously that the Kleisli construction for arrows corresponds precisely to a Freyd category. This turns the folklore claim that "arrows are Freyd categories", that has always remained informal, into a mathematically precise statement. The arrows-as-monoids perspective, however, is not so delicate as Freyd categories. Moreover it stresses the compositional infrastructure an arrow provides.

Ultimately, as with any denotational semantics, this approach aids functional programmers in reasoning about their programs. For example, it facilitates proving that the language extension induced by an arrow satisfies the desired domain-specific properties that initiated its design.

An interesting topic that has not yet been elaborated is recursion schemes for arrows (Erkök & Launchbury, 2002; Benton & Hyland, 2003), that might find a more thorough theoretical foundation in the present work.

Acknowledgement

We gratefully acknowledge John Power, who not only led us to the correspondence with Freyd categories, but also conjectured Lemma 6.1 immediately upon seeing Definition 6.5. We are also thankful for fruitful discussions with Thorsten Altenkirch, Tarmo Uustalu, and Paul Blain Levy.

A Coends

This appendix briefly recalls the notion of (parametrised) coends (for more information, see Mac Lane 1971, Section IX.6). This is then applied, as promised in section 4.4, to ensure that the category $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{V})$ has monoidal structure for suitable categories $\mathbf{C}$ and $\mathbf{V}$. We specialise the category $\mathbf{V}$ later.

A *coend* of a bifunctor $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$ consists of an object $V \in \mathbf{V}$ and a universal dinatural cocone $c : A \overset{\bullet}{\Longrightarrow} V$. Explicitly, a dinatural cocone c consists of morphisms $c_P : A(P,P) \to V$ for each $P \in \mathbf{C}$ such that, for each morphism $f : P \to Q$ of $\mathbf{C}$ the following diagram commutes.

$$
\begin{tikzcd}
 & A(P,P) \arrow[dr, "c_P"] & \\
A(P,Q) \arrow[ur, "A(\mathrm{id}_P, f)"] \arrow[dr, "A(f, \mathrm{id}_Q)"'] & & V \\
 & A(Q,Q) \arrow[ur, "c_Q"'] &
\end{tikzcd}
$$

Moreover c is universal in the sense that for every dinatural cocone $e : A \overset{\bullet}{\Longrightarrow} V'$ there is a unique mediating morphism $g : V \to V'$ as follows.

$$
\begin{tikzcd}
 & A(P,P) \arrow[dr, "c_P"] \arrow[drr, "e_P"] & & \\
A(P,Q) \arrow[ur, "A(\mathrm{id}_P, f)"] \arrow[dr, "A(f, \mathrm{id}_Q)"'] & & V \arrow[r, dashed, "g"] & V' \\
 & A(Q,Q) \arrow[ur, "c_Q"] \arrow[urr, "e_Q"'] & &
\end{tikzcd}
$$

The object V, when it exists, is unique up to isomorphism. By abuse of language, it is called *the coend* of A, and it is denoted by $V = \int^P A(P, P)$. The fact that coend V does not depend on the 'bound variable' P makes the remark in section 4.1 that the parameter P is auxiliary more specific.

We described for each $P, Q \in \mathbf{C}$ a coend as an object in $\mathbf{V}$. The following lemma shows that they cooperate so as to form a functor, even when parametrised, as desired later.

Lemma A.1
Let $\mathbf{V}$ be a monoidal category, and suppose $A, B : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$ are bifunctors such that for each $X, Y \in \mathbf{C}$, the functor

$$A(X, -) \otimes B(-, Y) : \mathbf{C} \times \mathbf{C}^{\mathrm{op}} \to \mathbf{V} \tag{A 1}$$

has a coend in $\mathbf{V}$. Then these coends extend to a functor

$$\int^P A(-, P) \otimes B(P, -) : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}. \tag{A 2}$$

Proof
This is a special case of Theorem IX.7.2 in (Mac Lane, 1971). □

Given two bifunctors $A, B : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$, we would like to define their monoidal product $A \otimes B : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$ to be the coending bifunctor (A 2). For this we need all the coends of (A 1) to exist. The naturality of *arr* in Lemma 4.2 suggests that exponentiation should be the unit of the desired monoidal structure. This requires that $\mathbf{C}$ be $\mathbf{V}$-enriched. This means that we assume objects $\mathbf{C}(X, Y) \in \mathbf{V}$ with suitable identity and composition morphisms $\mathcal{I}_X : I_{\mathbf{V}} \to \mathbf{C}(X, X)$ and $\mathcal{C}_{X,Y,Z} : \mathbf{C}(X, Y) \otimes \mathbf{C}(Y, Z) \to \mathbf{C}(X, Z)$. Moreover, we consider $\mathbf{V}$-bifunctors instead of bifunctors, which means that we also have morphisms $A_{(X,P),(Q,Y)} : \mathbf{C}(Q, X) \otimes \mathbf{C}(P, Y) \to A(Q, Y)^{A(X,P)}$ in $\mathbf{V}$ analogous to application of the bifunctor A. For more information, see (Borceux, 1994, Section 6.2; Kelly, 1982); another article with a lot of related information is (Cattani & Winskel, 2004). This requires $\mathbf{V}$ to have exponents (with respect to its tensors) and thus to be monoidal closed.

Indeed, under these conditions, the above ideas combined with Lemma A.1 provide the desired monoidal structure, as the next proposition demonstrates. Its construction dates back to (Day, 1970).

Proposition A.1
Let $\mathbf{V}$ be a symmetric monoidal closed category, and $\mathbf{C}$ a $\mathbf{V}$-enriched category. Suppose that all the coends of the functor (A 1) exist. Then $\mathbf{V}\text{-}\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{V})$ is monoidal.

Proof
Due to the $\mathbf{V}$-enrichedness of $\mathbf{C}$ and closedness of $\mathbf{V}$ we can define a $\mathbf{V}$-bifunctor $I : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$ by

$$I(X, Y) = \mathbf{C}(X, Y) \in \mathbf{V}$$

whose action on morphisms, *i.e.* the morphism

$$I_{(X,Y),(X',Y')} : \mathbf{C}(X', X) \otimes \mathbf{C}(Y, Y') \to \mathbf{C}(X', Y')^{\mathbf{C}(X,Y)}$$

in $\mathbf{V}$, is the transpose of the iterated composition morphism

$$\mathbf{C}(X',X)\otimes\mathbf{C}(Y,Y')\otimes\mathbf{C}(X,Y)\cong\mathbf{C}(X',X)\otimes\mathbf{C}(X,Y)\otimes\mathbf{C}(Y,Y')\to\mathbf{C}(X',Y').$$

Next, notice that Lemma A.1 duly enriches over $\mathbf{V}$ (see Kelly, 1982, Section 2.1), so that for $\mathbf{V}$-bifunctors $A,B:\mathbf{C}^{\mathrm{op}}\times\mathbf{C}\to\mathbf{V}$ we can define a $\mathbf{V}$-bifunctor $A\otimes B:\mathbf{C}^{\mathrm{op}}\times\mathbf{C}\to\mathbf{V}$ by

$$A\otimes B=\int^{P}A(-,P)\otimes B(P,-).$$

Recall that coend calculus is known to be associative up to isomorphism (Mac Lane, 1971, Proposition 9.8). So to show that the above are indeed a monoidal product and unit, it suffices to give natural isomorphisms $\lambda_A: I\otimes A\Rightarrow A$ and $\rho_A: A\otimes I\Rightarrow A$. We concentrate on ρ for the purposes of the proof. (In fact, this is an instance of the enriched Yoneda lemma.)

For each $P\in\mathbf{C}$ we define a morphism $e_P: I_{\mathbf{V}}\to A(X,Y)^{A(X,P)\otimes I(P,Y)}$ as the transpose of the following composite, where $I_{\mathbf{V}}$ is the monoidal unit of $\mathbf{V}$.

$$\begin{tikzcd}
 & A(X,Y) & \\
I_{\mathbf{V}}\otimes A(X,P)\otimes I(P,Y) \arrow[ur, dashed, "e'_P"] \arrow[d, "\cong"] & & A(X,P)\otimes A(X,Y)^{A(X,P)} \arrow[ul, "\mathrm{ev}^{\mathbf{V}}"'] \\
A(X,P)\otimes I_{\mathbf{V}}\otimes\mathbf{C}(P,Y) \arrow[rr, "\mathrm{id}\otimes\mathcal{I}_X\otimes\mathrm{id}"] & & A(X,P)\otimes\mathbf{C}(X,X)\otimes\mathbf{C}(P,Y) \arrow[u, "\mathrm{id}\otimes A_{(X,P),(X,Y)}"']
\end{tikzcd}$$

These form a $\mathbf{V}$-dinatural transformation $e: A(X,-)\otimes I(-,Y)\overset{\bullet}{\Longrightarrow}A(X,Y)$. (The fact that the domain of e_P is the monoidal unit in $\mathbf{V}$ is caused by the $\mathbf{V}$-enrichedness of the (di)natural transformation.) Since $A\otimes I$ is a coend, there is a unique morphism $\rho_{X,Y}:(A\otimes I)(X,Y)\to A(X,Y)$ such that $\rho_{X,Y}\circ c_P=e_P$, where $c: A(X,-)\otimes I(-,Y)\overset{\bullet}{\Longrightarrow}(A\otimes I)(X,Y)$ is the coending dinatural transformation. Then ρ is a natural isomorphism. The natural transformation whose component $\rho^{-1}_{X,Y}: I_{\mathbf{V}}\to(A\otimes I)(X,Y)^{A(X,Y)}$ at X,Y is the transpose of the composite

$$I_{\mathbf{V}}\otimes A(X,Y)\cong A(X,Y)\otimes I_{\mathbf{V}}\xrightarrow{\mathrm{id}\otimes\mathcal{I}_Y}A(X,Y)\otimes\mathbf{C}(Y,Y)\xrightarrow{c_Y}\mathrel{\mkern-14mu}\rightarrow(A\otimes I)(X,Y).$$

is the inverse of $\rho_{X,Y}$, which shows that ρ is indeed a natural isomorphism, as required. $\square$

To arrive at the monoidal structure of the previous proposition, we relied on the existence of all the coends of (A 1). However, since we are only after a monoid in $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}}\times\mathbf{C},\mathbf{V})$, it suffices to require the existence only of coends of the form $\int^{P}A(X,P)\times A(P,Y)$ for all $A:\mathbf{C}^{\mathrm{op}}\times\mathbf{C}\to\mathbf{C}$ and $X,Y\in\mathbf{C}$. Recall that in section 4.4 we only needed $\mathbf{C}$ to be small. The previous observations lead to the following extension of this requirement.

Definition A.1

A category $\mathbf{C}$ is said to *support arrows* in a symmetric monoidal closed category

$\mathbf{V}$ if it is $\mathbf{V}$-enriched, and for each $\mathbf{V}$-bifunctor $A : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{V}$ and each pair $X, Y \in \mathbf{C}$ the coend $\int^P A(X,P) \otimes A(P,Y)$ exists.

The previous machinery allows us to generalise our main definition, Definition 4.2.

Definition A.2
Let $\mathbf{C}$ be a category with finite products that supports arrows to $\mathbf{V}$. An *arrow* over $\mathbf{C}$ is a monoid in $\mathbf{V}\text{-}\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{V})$ that carries an internal strength.

Notable special cases of the previous definition are $\mathbf{V} = \mathbf{C}$ Cartesian closed, and $\mathbf{V} = \mathbf{Set}$. The latter case reduces the situation to profunctors, which we studied in Section 4.4. The former one most closely resembles categorical semantics of Haskell, in that it concerns just one category of types and terms. But if $\mathbf{C} = \mathbf{V}$ is to be Cartesian closed, small and cocomplete, then it is forced to be a preorder (Freyd, 1964, Chapter 3, Exercise D). However, small complete *internal* categories do exist (Hyland, 1988), and can indeed be used as models for polymorphic type theory. Working in such a universe is very similar to working in a polymorphic type theory as we have done in Section 4.4. Separating size issues is one of the reasons we have considered bifunctors to an enriching category $\mathbf{V}$ in this appendix.

B Bicategorical characterisation

Now that we have characterized arrows using categories enriched in a monoidal closed category in Appendix A, we may as well go one step further and give a unified characterisation of monads and arrows using categories enriched in a bicategory. As a bicategory is a 2-category in which composition is only associative up to isomorphism, this allows for the category of profunctors – after all, composition of profunctors depends on products in $\mathbf{Set}$, that are not strictly associative. This approach cleanly exhibits the motto

$$\frac{\text{monad}}{\text{arrow}} \cong \frac{\text{Functor}}{\text{Profunctor}}.$$

The definitions below clearly indicate that both arrows and monad are instances of monoids; the only difference is that one has to take the right category to base the monoid on. Finally, this unified approach gives an intuitive basis of Theorem 6.2.

Definition B.1
Let $\mathcal{V}$ be a bicategory, and $V \in \mathcal{V}$. By a *monoid on* V we mean a monoid in the monoidal category $\mathcal{V}(V,V)$, with the identity morphism and composition for monoidal structure.

To justify this terminology, observe that, an (ordinary) monoid in a monoidal category $\mathbf{C}$ is a monoid in the corresponding one-object bicategory.

Definition B.2
Let $\mathbf{C}$ be a category enriched in $\mathbf{V}$. A $\mathbf{V}$-monad on $\mathbf{C}$ is a monoid on $\mathbf{C}$ in $\mathbf{V}\text{-}\mathbf{Cat}$. A $\mathbf{V}$-arrow on $\mathbf{C}$ is a monoid on $\mathbf{C}$ in $\mathbf{V}\text{-}\mathbf{Prof}$.

For $\mathbf{C}$ a category, a $\mathbf{Set}$-monad on $\mathbf{C}$ is simply an (ordinary) monad. Unwinding the definition, a $\mathbf{Set}$-arrow A on $\mathbf{C}$ boils down to a monoid in $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}} \times \mathbf{C}, \mathbf{Set})$. Hence a $\mathbf{Set}$-arrow of the previous definition closely resembles Definition 4.2 for the case $\mathbf{V} = \mathbf{Set}$, since the only thing missing is the (internal) strength. Equivalently, we can see A as an 'index', and speak of a category $\mathbf{D}$ with the same objects as $\mathbf{C}$ and with homsets $\mathbf{D}(X,Y) = A(X,Y)$ instead, in which identity and composition are given by the monoidal structure on A. Still another equivalent way of putting this is an identity-on-objects functor $J : \mathbf{C} \to \mathbf{D}$. Conversely, given a category $\mathbf{D}$ and an identity-on-objects functor $J : \mathbf{C} \to \mathbf{D}$ we can reconstruct a $\mathbf{Set}$-arrow A on $\mathbf{C}$ by $A(X,Y) = \mathbf{D}(X,Y)$ and $A(f,g) = Jg \circ (-) \circ Jf$. This suggests that arrows over $\mathbf{C}$ should resemble identity-on-objects functors $\mathbf{C} \to \mathbf{D}$ to a category $\mathbf{D}$ with the same objects as $\mathbf{C}$, with added conditions corresponding to internal strength.

To incorporate the internal strength restriction, consider the following definition.

Definition B.3
Let $\mathbf{C}$ be a category with finite products. Define a $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$-enriched category $\mathrm{self}(\mathbf{C})$ by the same objects as $\mathbf{C}$, and homobjects $(\mathrm{self}(\mathbf{C}))(X,Y) = \mathbf{C}((-)\times X, Y)$.

For a category $\mathbf{C}$ with finite products, a $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$-monad on $\mathrm{self}(\mathbf{C})$ is an (ordinary) strong monad on $\mathbf{C}$. Analogously, we can talk about $\mathbf{Cat}(\mathbf{C}^{\mathrm{op}}, \mathbf{Set})$-arrows on $\mathrm{self}(\mathbf{C})$ as 'internally strong arrows over $\mathbf{C}$'. These correspond to a Freyd category $J : \mathbf{C} \to \mathbf{D}$.

C 2-categorical details in the Kleisli construction for arrows

The 2-category **Freyd** of Freyd categories is defined in the following obvious way.

- An object is a Freyd category $\mathbf{C} \xrightarrow{J} \mathbf{K}$.
- A 1-cell $(F,H) : (\mathbf{C} \xrightarrow{J} \mathbf{K}) \to (\mathbf{D} \xrightarrow{I} \mathbf{L})$ is a pair of a functor $F : \mathbf{C} \to \mathbf{D}$ preserving finite products and a functor $H : \mathbf{K} \to \mathbf{L}$ preserving premonoidal structures, such that $IF = HJ$.
- A 2-cell $(\alpha,\beta) : (F,H) \Rightarrow (F',H') : (\mathbf{C} \xrightarrow{J} \mathbf{K}) \to (\mathbf{D} \xrightarrow{I} \mathbf{L})$ is a pair of natural transformations $\alpha : F \Rightarrow F'$ and $\beta : H \Rightarrow H'$ such that $I\alpha = \beta J$.

The 2-category $\mathbf{Arr}_{\mathbf{Freyd}}$ of arrows on Freyd categories is as follows.

- An object is a pair $(\mathbf{C} \xrightarrow{J} \mathbf{K}, A)$ of a Freyd category an arrow A on that.
- A 1-cell $(F,H,\sigma) : (\mathbf{C} \xrightarrow{J} \mathbf{K}, A) \to (\mathbf{D} \xrightarrow{I} \mathbf{L}, B)$ is a 1-cell $(F,H) : (\mathbf{C} \xrightarrow{J} \mathbf{K}) \to (\mathbf{D} \xrightarrow{I} \mathbf{L})$ of **Freyd** together with a natural transformation

$$\begin{tikzcd} \mathbf{K}^{\mathrm{op}} \times \mathbf{K} \arrow[rr, "H^{\mathrm{op}} \times H"] \arrow[dr, "A"'] & \arrow[d, phantom, "\overset{\sigma}{\Longrightarrow}"] & \mathbf{L}^{\mathrm{op}} \times \mathbf{L} \arrow[dl, "B"] \\ & \mathbf{Set} & \end{tikzcd},$$

which is compatible with arr, $\ggg$ and $first$, in the following sense.

— $\sigma \bullet arr^A = \big(arr^B \circ (H^{\mathrm{op}} \times H)\big) \bullet \mathrm{Hom}(H)$. Here $\mathrm{Hom}(H)$ is the following natural transformation induced by H's action on morphisms.

$$\begin{tikzcd} \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \arrow[rr, "H^{\mathrm{op}} \times H"] \arrow[dr, "\mathrm{Hom}_{\mathbf{C}}"'] & \arrow[d, phantom, "\overset{\mathrm{Hom}(H)}{\Longrightarrow}"] & \mathbf{D}^{\mathrm{op}} \times \mathbf{D} \arrow[dl, "\mathrm{Hom}_{\mathbf{D}}"] \\ & \mathbf{Set} & \end{tikzcd} \tag{C 1}$$

— $\sigma \bullet {\ggg}^A = \big({\ggg}^B \circ (H^{\mathrm{op}} \times H)\big) \bullet (\sigma \otimes \sigma)$, where the natural transformation $\sigma \otimes \sigma$ is defined in the following obvious manner.

$$\begin{tikzcd} (A \otimes A)(X,Y) \arrow[r, dashed, "(\sigma\otimes\sigma)_{X,Y}"] & (B \otimes B)(HX,HY) \\ \coprod_{P \in \mathbf{C}} A(X,P) \times A(P,Y) \arrow[u, two heads] \arrow[r, dashed] & \coprod_{Q \in \mathbf{D}} B(HX,Q) \times B(Q,HY) \arrow[u, two heads] \\ A(X,P) \times A(P,Y) \arrow[u, "\kappa_P"] \arrow[r, "\sigma_{X,P} \times \sigma_{P,Y}"'] & B(HX,HP) \times B(HP,HY) \arrow[u, "\kappa_{HP}"'] \end{tikzcd}$$

Note that this construction of $\sigma \otimes \sigma : A \otimes A \Rightarrow (B \otimes B) \circ (H^{\mathrm{op}} \times H)$ is not an instance of the functoriality of $\otimes$.

— With $first$: for each X, Y, Z in $\mathbf{C}$,

$$\begin{tikzcd} A(X,Y) \arrow[rr, "\sigma"] \arrow[d, "first^A"'] & & B(HX,HY) \arrow[d, "first^B"] \\ A(X \boxtimes Z, Y \boxtimes Z) \arrow[r, "\sigma"'] & B\big(H(X \boxtimes Z), H(Y \boxtimes Z)\big) \arrow[r, "\cong"'] & B\big(HX \boxtimes HZ, HY \boxtimes HZ\big) \end{tikzcd}$$

where the isomorphism is because H preserves premonoidal structures.

- A 2-cell $(\alpha,\beta) : (F,H,\sigma) \Rightarrow (F',H',\sigma') : (\mathbf{C} \xrightarrow{J} \mathbf{K}, A) \to (\mathbf{D} \xrightarrow{I} \mathbf{L}, B)$ is a 2-cell $(\alpha,\beta) : (F,H) \Rightarrow (F',H')$ of **Freyd** which is compatible with σ and σ' in the following sense.

$$\begin{tikzcd} \mathbf{K}^{\mathrm{op}} \times \mathbf{K} \arrow[rr, bend left, "H^{\mathrm{op}} \times H'"] \arrow[rr, bend right] \arrow[rr, phantom, "H^{\mathrm{op}} \times \beta \Uparrow"] \arrow[dr, "A"'] & \arrow[d, phantom, "\overset{\sigma}{\Longrightarrow}"] & \mathbf{L}^{\mathrm{op}} \times \mathbf{L} \arrow[dl, "B"] \\ & \mathbf{Set} & \end{tikzcd} = \begin{tikzcd} \mathbf{K}^{\mathrm{op}} \times \mathbf{K} \arrow[rr, bend left, "H^{\mathrm{op}} \times H'"] \arrow[rr, bend right] \arrow[rr, phantom, "\beta^{\mathrm{op}} \times H' \Uparrow"] \arrow[dr, "A"'] & \arrow[d, phantom, "\overset{\sigma'}{\Longrightarrow}"] & \mathbf{L}^{\mathrm{op}} \times \mathbf{L} \arrow[dl, "B"] \\ & \mathbf{Set} & \end{tikzcd} . \tag{C 2}$$

Here, note that a natural transformation $\beta : H \Rightarrow H' : \mathbf{K} \to \mathbf{L}$ induces its dual $\beta^{\mathrm{op}} : H'^{\mathrm{op}} \Rightarrow H^{\mathrm{op}} : \mathbf{K}^{\mathrm{op}} \to \mathbf{L}^{\mathrm{op}}$, with the direction of $\Rightarrow$ reversed.

The 2-functor $Ins : \mathbf{Freyd} \to \mathbf{Arr}_{\mathbf{Freyd}}$ acts as follows.

- An object $\mathbf{C} \xrightarrow{J} \mathbf{K}$ is mapped to $(\mathbf{C} \xrightarrow{J} \mathbf{K}, \mathrm{Hom}_{\mathbf{K}})$. The bifunctor $\mathrm{Hom}_{\mathbf{K}}$ is obviously an arrow: its operation $first$ comes from the premonoidal structure of $\mathbf{K}$.
- A 1-cell (F,H) is mapped to $(F,H,\mathrm{Hom}(H))$. Compatibility of $\mathrm{Hom}(H)$ with $first$ is because H preserves premonoidal structures.
- A 2-cell $(\alpha,\beta) : (F,H) \Rightarrow (F',H')$ is mapped as it is. The compatibility of β with $\mathrm{Hom}(H)$ and $\mathrm{Hom}(H')$ amounts to the naturality of $\beta : H \Rightarrow H'$.

The 2-functor $\mathcal{K}\ell : \mathbf{Arr_{Freyd}} \to \mathbf{Freyd}$ is essentially the Kleisli construction for arrows in Definition 6.1. Namely,

- An object $(\mathbf{C} \stackrel{J}{\to} \mathbf{K}, A)$ is mapped to a Freyd category $\mathbf{C} \stackrel{J}{\to} \mathbf{K} \stackrel{J_A}{\to} \mathbf{K}_A$, where $\mathbf{K} \stackrel{J_A}{\to} \mathbf{K}_A$ is constructed like in Definition 6.1. The composition $J_A \circ J$ is indeed a Freyd category: for example it preserves central morphisms essentially due to the arrow law (6').
- A 1-cell $(F, H, \sigma) : (\mathbf{C} \stackrel{J}{\to} \mathbf{K}, A) \to (\mathbf{D} \stackrel{I}{\to} \mathbf{L}, B)$ induces a functor $\tilde{H} : \mathbf{K}_A \to \mathbf{L}_B$ such that: $\tilde{H}X = HX$ on objects and $\tilde{H}f = \sigma_{X,Y}(f)$ on morphisms.
- A 2-cell (α, β) is mapped to $(\alpha, \tilde{\beta})$, where a component $\tilde{\beta}_X$ is given by $arr(\beta_X)$. The naturality of $\tilde{\beta}$ amounts to the coherence condition (C 2).

References

Alimarine, A., Smetsers, S., van Weelden, A., van Eekelen, M., & Plasmeijer, R. (2005). There and back again: arrows for invertible programming. *Pages 86–97 of: Proceedings of the 2005 ACM SIGPLAN workshop on Haskell.* ACM Press.

Bénabou, J. (2000). *Distributors at work.* Lecture notes written by Thomas Streicher.

Benton, N., & Hyland, M. (2003). Traced premonoidal categories. *Theoretical informatics and applications*, **37**(4), 273–299.

Borceux, F. (1994a). *Handbook of categorical algebra.* Encyclopedia of Mathematics and its applications, vol. 1. Cambridge University Press.

Borceux, F. (1994b). *Handbook of categorical algebra.* Encyclopedia of Mathematics and its applications, vol. 2. Cambridge University Press.

Cattani, G. L., & Winskel, G. 2004 (October). *Profunctors, open maps and bisimulation.* Tech. rept. RS-04-22. BRICS, Aarhus, Denmark.

Day, B. J. (1970). *Reports of the midwest category seminar.* Lecture Notes in Mathematics, vol. 137. Springer. Chap. On Closed Categories of Functors, pages 1–38.

Erkök, L., & Launchbury, J. (2002). A recursive do for haskell. *Pages 29–37 of: Proceedings of the 2002 ACM SIGPLAN workshop on Haskell.* ACM Press.

Freyd, P. J. (1964). *Abelian categories: An introduction to the theory of functor.* New York: Harper and Row.

Heunen, C., & Jacobs, B. (2006). Arrows, like monads, are monoids. *Pages 219–236 of:* Brookes, S., & Mislove, M. (eds), *Mathematical foundations of programming semantics, twenty-second annual conference.* Electronic Notes in Theoretical Computer Science, vol. 158.

Hughes, J. (2000). Generalising monads to arrows. *Science of computer programming*, **37**, 67–111.

Hughes, J. (2005). *Advanced functional programming.* Lecture Notes in Computer Science, vol. 3622. Springer. Chap. Programming with Arrows, pages 73–129.

Hyland, J. M. E. (1988). A small complete category. *Annals of pure & applied logic*, **40**, 135–165.

Jacobs, B. (1999). *Categorical logic and type theory.* North Holland.

Jacobs, B. (2006). Distributive laws for the coinductive solution of recursive equations. *Information and computation*, **204**(4), 561–587.

Jacobs, B., & Hasuo, I. (2006). Freyd is kleisli, for arrows. McBride, C., & Uustalu, T. (eds), *Workshop on mathematically structured functional programming (MSFP 2006).* Electronic Workshops in Computing (eWiC).

Kelly, G. M. (1982). *Basic concepts of enriched category theory.* London Mathematical Society Lecture Notes, vol. 64. Cambridge University Press.

Levy, P. B., Power, A. J., & Thielecke, H. (2003). Modelling environments in call-by-value programming languages. *Information and computation*, **185**, 182–210.

Li, P., & Zdancewic, S. (2008). *Arrows for secure information flow.* Tech. rept. MS-CIS-08-02. University of Pennsylvania.

Mac Lane, S. (1971). *Categories for the working mathematician.* Springer.

Moggi, E. (1989). Computational lambda-calculus and monads. *Logic in computer science.*

Paterson, R. (2001). A new notation for arrows. *Pages 229–240 of: International conference on functional programming.* ACM Press.

Paterson, R. (2003). Arrows and computation. *Pages 201–222 of:* Gibbons, J., & de Moor, O. (eds), *The fun of programming.* Palgrave.

Peyton Jones, S.L. (ed). (2003). *Haskell 98 language and libraries: the revised report.* Cambridge University Press.

Power, J., & Thielecke, H. (1997). Environments, continuation semantics and indexed categories. *Pages 391–414 of:* Abadi, M., & Ito, T. (eds), *Theoretical aspects of computer software.* Lectures Notes in Computer Science, no. 1281. Springer, Berlin.

Robinson, E.P., & Power, J. (1997). Premonoidal categories and notions of computation. *Mathematical structures in computer science*, **7**(5), 453–468.

Street, R. (1972). The formal theory of monads. *Journal of pure & applied algebra*, **2**, 149–169.

Swierstra, S. D., & Duponcheel, L. (1996). Deterministic, error-correcting combinator parsers. *Pages 184–207 of:* Launchbury, J., Meijer, E., & Sheard, T. (eds), *Advanced functional programming.* Lectures Notes in Computer Science, vol. 1129. Springer, Berlin.

Uustalu, T., & Vene, V. (2005). Signals and comonads. *Journal of universal computer science*, **11**(7), 1310–1326".

Vizzotto, J. K., Altenkirch, T., & Sabry, A. (2006+). Structuring quantum effects: Superoperators as arrows. *to appear in mathematical structures in computer science, special issue on quantum programming languages.*

Wadler, P. (1992). Monads for functional programming. *Program design calculi: Proceedings of the 1992 marktoberdorf international summer school.* Springer-Verlag.