Modelling environments in call-by-value programming languages

Paul Blain Levy
University of Birmingham, Birmingham B15 2TT, UK
pbl@cs.bham.ac.uk

and

John Power[1]
University of Edinburgh, Edinburgh EH9 3JZ, UK
ajp@dcs.ed.ac.uk

and

Hayo Thielecke
University of Birmingham, Birmingham B15 2TT, UK
H.Thielecke@cs.bham.ac.uk

Version: 15 November 2002

In categorical semantics, there have traditionally been two approaches to modelling environments, one by use of finite products in cartesian closed categories, the other by use of the base categories of indexed categories with structure. Each requires modifications in order to account for environments in call-by-value programming languages. There have been two more general definitions along both of these lines: the first generalising from cartesian to symmetric premonoidal categories, the second generalising from indexed categories with specified structure to κ-categories. In this paper, we investigate environments in call-by-value languages by analysing a fine-grain variant of Moggi's computational λ-calculus, giving two equivalent sound and complete classes of models: one given by closed Freyd categories, which are based on symmetric premonoidal categories, the other given by closed κ-categories.

1. INTRODUCTION

Traditionally in denotational semantics, there have been two categorical ways of modelling environments. The first is given by finite products in a cartesian closed category, as for instance in modelling the simply typed λ-calculus. Over the years, that has gradually been extended. For instance, in order to model partiality, one must generalise from finite product structure to symmetric monoidal structure; and that has been further generalised to the notion of symmetric premonoidal structure [12].

A premonoidal category is essentially a monoidal category except that the tensor need only be a functor in two variables separately, and not necessarily a bifunctor: given maps $f : A \to A'$ and $g : B \to B'$, the evident two maps from $A \otimes B$ to $A' \otimes B'$

[1] This work is supported by EPSRC grant GR/J84205: Frameworks for programming language semantics and logic.

may differ. Such structures arise naturally in the presence of computational effects, where the difference between these two maps is a result of sensitivity to evaluation order. So that is the structure we need in order to model environments in the presence of continuations or other such strong computational effects. A program phrase in environment Γ is modelled by a morphism in the premonoidal category with domain $[\![\Gamma]\!]$.

The second approach to modelling environments categorically, also used to model the simply typed λ-calculus, is based on indexed categories with structure (for an application to type theory, see [4]). The idea is that contexts are indices for the categories in which the terms definable in that context are modelled. Here, a program phrase in context Γ is modelled by an element $\mathbf{1} \longrightarrow [\![A]\!]$ in a category that implicitly depends on Γ, that is, by an arrow from $\mathbf{1}$ to $[\![A]\!]$ in the fibre of the indexed category over $[\![\Gamma]\!]$. We consider a weak version of indexed category with structure, called a κ-category, inspired by some work by Masahito Hasegawa [3].

The main result of this paper is to prove the above two models of environments equivalent. More precisely, we show that every symmetric premonoidal category with a little more of the structure cited above, gives rise to a κ-category, and that this gives an equivalence between the classes of symmetric premonoidal categories with such structure and κ-categories. The extra structure we need on a symmetric premonoidal category $\mathcal{K}$ is a category with finite products $\mathcal{C}$, whose objects are the same as $\mathcal{K}$, and an identity on objects structure-preserving functor $J : \mathcal{C} \longrightarrow \mathcal{K}$: we call these *Freyd categories.* (They are called *value/producer structures* in [8].)

We then refine the notion κ-category to an equivalent but more useful structure called *strong κ-category.* Indeed, it could be said that the main advantage of the former is as a stepping-stone to the latter. The flexibility present in the definition of strong κ-category allows us to present call-by-value semantics in a way that, on the one hand, is computationally appropriate, but, on the other hand, is mathematically convenient. We will use the example of continuations to illustrate this.

Having established an equivalence between these various ways of modelling environments, we extend that equivalence to study the modelling of higher order structure. It is not as simple as asking for a routine extension of the notion of closedness from that for a cartesian category to a premonoidal category, as one usually considers λ-terms as values, and we distinguish between values and ordinary terms (called producers). This leads us to a notion of closedness for a Freyd category [11]. So we extend our equivalence between Freyd categories and κ-categories to one between closed Freyd categories and closed κ-categories, and likewise for strong κ-categories.

For concreteness, we shall study the modelling of environments in call-by-value languages with computational effects by studying models of a fine-grain form of Moggi's computational λ-calculus [9] (also known as λ_{c}-calculus). Moggi's calculus was specifically designed as a variant of the simply typed λ-calculus apposite for the study of computational effects. It is a natural fragment of a call-by-value programming language such as ML. Its models were defined to be λ_{c}-models, which consist of a cartesian category (i.e. category with distinguished terminal object and binary products) $\mathcal{C}$, and a strong monad $\mathcal{T}$ on $\mathcal{C}$, and *Kleisli exponentials* i.e. for each $B, C \in \mathbf{Ob}\,\mathcal{C}$ an isomorphism

$$\mathcal{C}(A \times B, TC) \cong \mathcal{C}(A, B \to_{\mathsf{Kl}} C) \text{ natural in } A$$

for some specified object $B \to_{\mathsf{Kl}} C$. The class of λ_{c}-models is sound and complete for the calculus, but it does not provide direct models in that a term of type τ in context Γ is not modelled by an arrow in $\mathcal{C}$ from the semantics of Γ to the semantics of τ, but by a derived construction in terms of the monad. We give equivalent formulations of λ_{c}-models providing a more direct semantics, in terms of closed Freyd categories, closed κ-categories and closed strong κ-categories.

This paper is an extension of our work in [13], incorporating some results of [14].

1.1. Related Work

The relationship between Freyd categories and κ-categories is related to work by Blute, Cockett, and Seely [2]. Implicit in their work is the construction that, to a Freyd category, assigns a κ-category. The latter are closely related to their context categories. Identifying precisely which indexed categories thus arise did not appear in their work. Although their work was not mainly directed towards the same problems as ours, it is interesting to note that the type theory suggested there is quite different to that presented here.

A treatment of the categorical semantics for the new *call-by-push-value* paradigm [7] is given in Levy's thesis [8]. Like our treatment here, it essentially presents three approaches: strong monads, Freyd categories and indexed categories.

1.2. Overview

The paper is organised as follows. In Sect. 2, we motivate by some examples why one seeks a more general notion than that of monoidal category for environments in call-by-value programming languages. We then define fine-grain call-by-value in Sect. 3. In Sect. 4, we define Freyd categories and show how the first-order fragment of our calculus can be interpreted in them. Soundness and completeness of this semantics is proved in Sect. 5. In Sect. 6, we define the notion of κ-category, and establish the relationship between κ-categories and Freyd categories. We extend our equivalence to one incorporating higher order structure in Sect. 7. Appendix A recalls the definitions of premonoidal category and related structures.

2. SOME EXAMPLES

The tuple (similarly, list) notation present in many call-by-value programming languages such as ML or Scheme may, at first sight, suggest that the appropriate semantic setting ought to be a cartesian or at least monoidal category.

But in terms of evaluation in a call-by-value language, a tuple (M, N) means that each component has to be evaluated.

This can be made explicit by naming the intermediate values. If the first component is to be evaluated first, one would write:

$$\texttt{let x} = M \texttt{ in let y} = N \texttt{ in } (\texttt{x}, \texttt{y})$$

Alternatively, to evaluate the second component first, one writes:

$$\texttt{let y} = N \texttt{ in let x} = M \texttt{ in } (\texttt{x}, \texttt{y})$$

The `let`-notation, then, has the advantage that the implicit sequencing is made explicit in the textual representation. Clearly, it would be a disadvantage to make irrelevant sequencing information explicit, but in examples such as those below, the sequencing information is *vital*, so must be made explicit.

For example, in a language with state, there are two possible meanings of a tuple (M, N), depending on which component is evaluated first. Consider the following examples, where we make the evaluation order explicit by using `let`.

EXAMPLE 2.1.
```
let val s = ref 0 in
  let val x = (s := !s + 1; !s) in
    let val y = (s := !s + 1; !s) in
      #1(x,y)
    end
  end
end;

let val s = ref 0 in
  let val y = (s := !s + 1; !s) in
    let val x = (s := !s + 1; !s) in
      #1(x,y)
    end
  end
end;
```

EXAMPLE 2.2. Just as for state, in the presence of continuations (first-class or otherwise) there are two possible meanings of the tuple `(throw k 1, throw k 2)`.

```
callcc(fn k =>
  let val x = throw k 1 in
    let val y = throw k 2 in
      #1(x,y)
    end
  end);

callcc(fn k =>
  let val y = throw k 2 in
    let val x = throw k 1 in
      #1(x,y)
    end
  end);
```

If this were to be interpreted in a monoidal category directly with the tupling notation, one could not distinguish between the two composites. The problem is that, in a monoidal category, given maps $f : A \longrightarrow A'$ and $g : B \longrightarrow B'$, the two induced maps from $A \otimes B$ to $A' \otimes B'$ are equal. This makes monoidal categories suitable for those cases where both composites are evaluated in parallel *or* where there can be no interference between the two (which would be the case, say, if both had access to disjoint pieces of state). But with control, as given by continuations, we have both a sequential evaluation order and interference between the components, since a jump in one will prevent the other from being evaluated at all.

Put differently, the presence of computational effects, like state and control, "breaks" the bifunctoriality, so one is left with a binoidal category as defined in the appendix.

Adding higher order structure to this analysis, it follows that there is delicacy in modifying the simply typed λ-calculus in order to provide a variant that is suitable for the study of call-by-value languages with the possibility of computational effects such as state or continuations. Such a language was provided by Moggi's computational λ-calculus, or λ_{c}-calculus [9], a variant of which which we analyse in this paper. A key point in modelling the λ_{c}-calculus is that, as explained above, one needs care in modelling environments.

3. FROM COARSE-GRAIN TO FINE-GRAIN CALL-BY-VALUE

3.1. λ_{c} Calculus And Monadic Metalanguage

First we will recall λ_{c}-calculus and then explain why we choose to work with a more fine-grain language.

We will consider the following types only:

$$A ::= \quad 1 \mid A \times A \mid A \to A$$

This does not mean that there cannot be other type constructors such as $+$ and `bool`—after all, a language without `bool` would not be of much use—but rather that 1, $\times$ and $\to$ are the only type constructors whose categorical semantics we address in this paper. Furthermore, we will omit all rules, equations etc. for 1 as they are directly analogous to those for $\times$.

We first give the λ-calculus constructs for these type:

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash \mathtt{x} : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma, \mathtt{x} : A \vdash N : B}{\Gamma \vdash \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ M.\ N : B}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M' : A'}{\Gamma \vdash (M, M') : A \times A'} \qquad \frac{\Gamma \vdash M : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash N : B}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N : B}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash M : B}{\Gamma \vdash \lambda\mathtt{x}.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

We make some comments about these constructs:

- While the declaration construct `let` is technically redundant because `let x be` $M.\ N$ is equivalent to $(\lambda\mathtt{x}.N)M$, we prefer to include it as primitive because we feel it is more basic than the $\to$ constructs.
- The elimination rule we have given for $A \to B$ is a pattern-match construct (`pm` is an abbreviation for "pattern-match"). We could alternatively have used projection constructs.
- Here and throughout the paper, in order to reduce clutter, we omit type subscripts on bindings of identifiers, but strictly speaking they should be present.

The λ_{c} calculus is an equational theory for the above constructs providing equations that hold as observational equivalences when we add computational effects and impose a call-by-value operational semantics. To formulate it, an auxiliary predicate $\downarrow$ on terms is required, where $V \downarrow$ means that V is effect-free in any environment. We call such a term a *value* (although this is not quite consistent with the operational notion of value), and we call a general term a *producer* because (in a given environment) it produces a value. This predicate is inductively given by

$$V ::= \ \mathtt{x} \mid (V,V) \mid \lambda\mathtt{x}.M \mid \mathtt{let\ x\ be}\ V.\ V \mid \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).V$$

where V ranges over values and M ranges over producers. We can then provide axioms for the λ_{c}-calculus such as the β-value law

$$(\lambda\mathtt{x}.M)V = M[V/\mathtt{x}]$$

where V ranges over values and M ranges over producers.

This calculus has many models. As a leading example, consider the semantics for global store, where S is the set of stores. Each type A, and hence each context Γ, denotes a set. A producer $\Gamma \vdash M : A$ denotes a function from $S \times [\![\Gamma]\!]$ to $S \times [\![A]\!]$—we will call this function $[\![M]\!]^{\mathsf{prod}}$. If $M \downarrow$, then M additionally denotes a function from $[\![\Gamma]\!]$ to $[\![A]\!]$—we will call this function $[\![M]\!]^{\mathsf{val}}$.

λ_{c}-calculus has proved enormously helpful in analyzing semantics of call-by-value. However, it has some problems.

1. The theory is not purely equational, because the predicate $\downarrow$ is required.

2. The choices we made, that an application MN should be evaluated operator first and that a pair (M,N) should be evaluated left-to-right, are quite arbitrary.

3. Application and pairing are clearly complex constructs. Here, for example, is the semantics of application:

   $$[\![MN]\!]\rho s = \mathsf{pm}\ [\![M]\!]\ \mathtt{as}\ (s',f).\mathsf{pm}\ [\![N]\!]s'\rho\ \mathtt{as}\ (s'',a).f(s'',a)$$

   It clearly reflects the 3-stage process of evaluating MN: first evaluate M to $\lambda\mathtt{x}.M'$, then evaluate N to V, then evaluate $M'[V/\mathtt{x}]$.

4. An effect-free term $M \downarrow$ has two denotations $[\![M]\!]^{\mathsf{prod}}$ and $[\![M]\!]^{\mathsf{val}}$, within the same model. They are related, as we have $[\![M]\!]^{\mathsf{prod}}(s,\rho) = (s, [\![M]\!]^{\mathsf{val}}\rho)$, but nevertheless we would prefer them to be the denotations of syntactically distinguished terms, so that each term has just one denotation.

Moggi resolved all these problems simultaneously in [10] by providing another language, the *monadic metalanguage*, from which they are all absent and into which λ_{c}-calculus can be translated. In the monadic metalanguage a term $\Gamma \vdash M : A$ denotes, in the global store model, a function from $[\![\Gamma]\!]$ to $[\![A]\!]$. Thus it can be said that whereas in the λ_{c}-calculus a general term is a producer, in the monadic metalanguage a general term is a value. A producer then has to be represented as a term $\Gamma \vdash M : TA$ where TA would be written $1 \to A$ in the types that we are using.

But there is a disadvantage to the monadic metalanguage, as compared with the λ_{c}-calculus: it is not easy to formulate operational semantics. As an example, consider λ_{c}-calculus with global store constructs: we can easily give an inductive definition of big-step semantics in the form $s, M \Downarrow s', V$ where M is a closed producer of any type, and V is a closed value of the same type. But for monadic metalanguage with global store constructs, it is not clear what form the big-step semantics should take.

3.2. Fine-Grain Call-By-Value

We explained in the previous section that

- Moggi's λ_c-calculus is a language of *producers*, in which a term $\Gamma \vdash M : A$ denotes (in the global store model) a function from $S \times [\![\Gamma]\!]$ to $S \times [\![A]\!]$
- Moggi's monadic metalanguage is a language of *values*, in which a term $\Gamma \vdash M : A$ denotes (in the global store model) a function from $[\![\Gamma]\!]$ to $[\![A]\!]$.

We now present *fine-grain call-by-value* that has two separate judgements for values and producers: we write $\Gamma \vdash^{\mathsf{v}} V : A$ to say that V is a value and we write $\Gamma \vdash^{\mathsf{p}} M : A$ to say that M is a producer. As a result of having these two judgements, it combines the advantages of λ_c-calculus (suitability for operational semantics) with the advantages of the monadic metalanguage (constructs are canonical and semantically simple, theory is purely equational, terms have just one denotation within a given model).

Corresponding to our two-part development of the categorical semantics, we present fine-grain call-by-value in two parts: the first order fragment presented in Fig. 1 and the higher order constructs presented in Fig. 2. We repeat that there can be other type constructors besides $1, \times, \rightarrow$, such as `bool` or $+$, but it is only the former whose categorical semantics we are studying; also that we omit constructs and equations for 1 because they are analogous to those for $\times$.

The key producer terms are these:

- `produce` V is the *trivial producer*: the construct `produce` explicitly converts the value V into a producer, unlike in λ_c-calculus where this conversion is invisible. `produce` is similar to `return` in Pascal and Java.
- M `to x.` N is the *sequenced producer*: it means "execute M, bind `x` to the value it returns, then execute N".

We represent the producers of λ_c-calculus in fine-grain CBV as follows:

λ_c-calculus producer	fine-grain CBV producer
$\mathtt{x}$	$\mathtt{produce\ x}$
$\lambda\mathtt{x}.M$	$\mathtt{produce}\ \lambda\mathtt{x}.M$
$\mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).N$	$M\ \mathtt{to\ w.\ pm\ w\ as}\ (\mathtt{x},\mathtt{y}).N$
$\mathtt{let\ x\ be}\ M.\ N$	$M\ \mathtt{to\ w.\ let\ x\ be\ w.}\ N$
(M,N)	$M\ \mathtt{to\ x.}\ N\ \mathtt{to\ y.\ produce}\ (\mathtt{x},\mathtt{y})$
MN	$M\ \mathtt{to\ f.}\ N\ \mathtt{to\ x.\ fx}$

This transform makes it clear that in λ_c-calculus, the application construct does more than just application, the declaration construct `let` does more than just declaration, and so forth. (It is to make this point clear that we translate `let x be` $M.\ N$ as shown rather than as the shorter M `to x.` N.) Sequencing and producing are hidden inside the λ_c constructs, and fine-grain CBV makes them explicit (just as the monadic metalanguage does).

Notice how the translation of (M, N) makes the order of evaluation apparent—if we wanted right-to-left evaluation order we would translate it as

$$N\ \mathtt{to\ y.}\ M\ \mathtt{to\ x.\ produce}\ (\mathtt{x},\mathtt{y})$$

Types

$$A ::= \quad 1 \mid A \times A$$

Judgements

$\Gamma \vdash^{\mathsf{v}} V : A$	V is a *value* of type A
$\Gamma \vdash^{\mathsf{p}} M : A$	M is a *producer* of type A

Terms

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ W : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{\Gamma \vdash^{\mathsf{p}} \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} (V, V') : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{p}} M : B}{\Gamma \vdash^{\mathsf{p}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{p}} \mathtt{produce}\ V : A} \qquad \frac{\Gamma \vdash^{\mathsf{p}} M : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} N : B}{\Gamma \vdash^{\mathsf{p}} M\ \mathtt{to}\ \mathtt{x}.\ N : B}$$

Equations, using convention at end of Sect. 3.2

$$\begin{array}{lccc}
(\beta) & \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ W & = & W[V/\mathtt{x}] \\
(\beta) & \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M & = & M[V/\mathtt{x}] \\
(\beta) & \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W & = & W[V/\mathtt{x}, V'/\mathtt{y}] \\
(\beta) & \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M & = & M[V/\mathtt{x}, V'/\mathtt{y}] \\
(\beta) & \mathtt{produce}\ V\ \mathtt{to}\ \mathtt{x}.\ M & = & M[V/\mathtt{x}] \\
(\eta) & W[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W[(\mathtt{x}, \mathtt{y})/\mathtt{z}] \\
(\eta) & M[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M[(\mathtt{x}, \mathtt{y})/\mathtt{z}] \\
(\eta) & M & = & M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{produce}\ \mathtt{x} \\
 & (M\ \mathtt{to}\ \mathtt{x}.\ N)\ \mathtt{to}\ \mathtt{y}.\ P & = & M\ \mathtt{to}\ \mathtt{x}.\ (N\ \mathtt{to}\ \mathtt{y}.\ P)
\end{array}$$

FIG. 1 First Order Fragment Of Fine-Grain CBV

Higher Order Types

$$A ::= \quad \cdots \mid A \to A$$

Higher Order Terms

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{\Gamma \vdash^{\mathsf{v}} \lambda \mathtt{x}.M : A \to B} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \to B \quad \Gamma \vdash^{\mathsf{v}} W : A}{\Gamma \vdash^{\mathsf{p}} VW : B}$$

Higher Order Equations, using convention at end of Sect. 3.2

$$\begin{array}{lrcl} (\beta) & (\lambda \mathtt{x}.M)V & = & M[V/\mathtt{x}] \\ (\eta) & V & = & \lambda \mathtt{x}.(Vx) \end{array}$$

FIG. 2 Higher Order Constructs Of Fine-Grain CBV

This is meant to emphasise that, in a *pre*monoidal setting, we need to take a little more care than in a monoidal one (where the left-to-right and right-to-left translations would be equivalent).

Each equation is presented subject to the convention that if a term M (more accurately, a metasyntactic identifier ranging over terms) occurs in the scope of an $\mathtt{x}$-binder and also occurs not in the scope of an $\mathtt{x}$-binder then $\mathtt{x}$ must not be in the context of M. For example, in the η-law in Fig. 2, this convention implies that $\mathtt{x}$ must not be in the context of V.

4. FREYD-CATEGORIES

In this Section, we define Freyd categories.

For reference, we include precise definitions of premonoidal category [12] and related structures in the appendix. To complete the category theory required to formulate the semantics, we say

DEFINITION 4.1. A *Freyd category* consists of a cartesian category $\mathcal{C}$, a symmetric premonoidal category $\mathcal{K}$ with the same objects as $\mathcal{C}$, and an identity on objects functor $J : \mathcal{C} \longrightarrow \mathcal{K}$, preserving strict symmetric premonoidal structure, whose image lies inside the centre of $\mathcal{K}$. $\mathcal{C}$ is called the *value category* and its morphisms are called *value morphisms*. $\mathcal{K}$ is called the *producer category* and its morphisms are called *producer morphisms*.

We write $\times$ rather than $\otimes$ for the binary operation on objects provided by the premonoidal structure, because it is a product operation in $\mathcal{C}$. The interpretation of the first order fragment in a Freyd category is organized as follows.

- A type denotes an object in the obvious way.
- The context $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$ denotes the object $[\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$.
- A value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a value morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$.
- A producer $\Gamma \vdash^{\mathsf{p}} M : A$ denotes a producer morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$.

For example:

- if $\Gamma \vdash^{\mathsf{v}} V : A$ then $\mathtt{produce}\ V$ denotes $J[\![V]\!]$

- if $\Gamma \vdash^{\mathsf{p}} M : A$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} N : B$ then M $\mathtt{to}$ $\mathtt{x}.$ N denotes

$$[\![\Gamma]\!] \xrightarrow{J(\mathsf{id},\mathsf{id})} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{[\![\Gamma]\!] \times [\![M]\!]} [\![\Gamma]\!] \times [\![A]\!] \xrightarrow{[\![N]\!]} [\![B]\!]$$

Corresponding to the examples of state and control (Examples 2.1 and 2.2), we sketch how each of these computational effects gives rise to a Freyd category.

Example 4.2. This example provides semantics for global store, and corresponds to Moggi's $S \to (S \times -)$ strong monad. First fix a set S. Then let $\mathcal{C}$ be **Set**, and let $\mathcal{K}$ be the category in which an object is a set and a morphism from X to Y is a function from $S \times X$ to $S \times Y$, with the evident identity and composition. J is defined in the evident way.

Example 4.3. This example provides semantics for control effects that manipulate continuations, and corresponds to Moggi's $(- \to \mathsf{Ans}) \to \mathsf{Ans}$ strong monad. First fix a set Ans. Then let $\mathcal{C}$ be **Set** and let $\mathcal{K}$ be the category in which an object is a set and a morphism from X to Y is a function from $X \times (Y \to \mathsf{Ans})$ to Ans, with the evident identity and composition. J is defined in the evident way.

To define *closed* Freyd category, just as for cartesian closed categories, we make use of "representable functors":

Definition 4.4. Let $\mathcal{F}$ be a functor from $\mathcal{C}^{\mathrm{op}}$ to **Set**. A *representation* for $\mathcal{F}$ consists of an object V (the *vertex*) together with an isomorphism

$$\mathcal{F}X \cong \mathcal{C}(X, V) \qquad\qquad \text{natural in } X.$$

Here is a well-known example.

Definition 4.5. 1. Let A and B be objects in a cartesian category $\mathcal{C}$. An *exponential* from A to B is a representation for the functor $\lambda X.\mathcal{C}(X \times A, B)$. Explicitly, this is an object V (the vertex) together with an isomorphism

$$\mathcal{C}(X \times A, B) \cong \mathcal{C}(X, V) \qquad\qquad \text{natural in } X.$$

2. Let (T, η, μ, t) be a strong monad on a cartesian category $\mathcal{C}$. To give *Kleisli exponentials* for this monad means to give an exponential from A to TB for each pair of objects A, B.

Definition 4.6. A *closed Freyd category* consists of a Freyd category together with, for each pair of objects A, B a representation for the functor $\lambda X.\mathcal{K}(J(X \times A), B)$, whose vertex we call $A \to B$. Explicitly, this gives an isomorphism

$$\mathcal{K}(J(X \times A), B) \cong \mathcal{C}(X, A \to B) \qquad\qquad \text{natural in } X \in \mathcal{C}^{\mathrm{op}}.$$

We can shorten this definition by recalling that we can define a *right adjoint* for a functor $F : \mathcal{C} \longrightarrow \mathcal{K}$ to consist of, for each object B of $\mathcal{K}$, a representation for the functor $\lambda X.\mathcal{K}(FX, B)$. Consequently, we can define a closed Freyd category to consist of a Freyd category, together with, for each object A, a right adjoint for the functor $J(- \times A) : \mathcal{C} \longrightarrow \mathcal{K}$. In particular, the functor $J : \mathcal{C} \longrightarrow \mathcal{K}$ will have a right adjoint, and so, because J is identity-on-objects, $\mathcal{K}$ is the Kleisli category for a monad on $\mathcal{C}$. A variant of one of the main theorems of [11] is

THEOREM 4.7. *To give a closed Freyd category is equivalent to giving a λ_{c}-model. More precisely, the 2-category of closed Freyd categories and the 2-category of λ_{c}-models, as defined in Appendix C, are 2-equivalent.*

That is as good a result as one can imagine to relate closed Freyd categories with strong monads. The theorem shows that closed Freyd categories are equivalent to Moggi's λ_{c}-models, so (as we shall see in Sect. 5) form a sound and complete class of models for fine-grain call-by-value.

In the light of this result, the reader may wonder what advantage Freyd categories have over strong monads with Kleisli exponentials. The answer is that the former provides greater flexibility in the organization of a model. For example, when organizing a global store model with a strong monad, a producer $\Gamma \vdash^{\mathsf{p}} M : A$ must denote a function from $[\![\Gamma]\!]$ to $S \to (S \times [\![A]\!])$. When organizing the model as a Freyd category, we can still interpret M in this way if we choose, but we also have the option of using the Freyd category in Example 4.2 so that M denotes a function from $S \times [\![\Gamma]\!]$ to $S \times [\![A]\!]$. This is computationally appropriate: it says that M, when executed in a given store $s \in S$ and environment $\rho \in [\![\Gamma]\!]$, terminates in a store $s' \in S$ when it produces a value $a \in [\![A]\!]$.

For another example, when organizing a continuation model with a strong monad, a producer $\Gamma \vdash^{\mathsf{p}} M : A$ must denote a function from $[\![\Gamma]\!]$ to $([\![A]\!] \to \mathsf{Ans}) \to \mathsf{Ans}$. When organizing the model as a Freyd category, we can still interpret M in this way if we choose, but we also have the option of using the Freyd category in Example 4.3 so that M denotes a function from $[\![\Gamma]\!] \times ([\![A]\!] \to \mathsf{Ans})$ to Ans. This is computationally appropriate: it says that M, when executed in a given environment $\rho \in [\![\Gamma]\!]$ and current continuation $K \in [\![A]\!] \to \mathsf{Ans}$, gives a final answer in Ans.

5. SOUNDNESS AND COMPLETENESS

We will formulate soundness and completeness results relating the first order fragment presented in Fig. 1 to Freyd categories. There are analogous results relating the whole language presented in Fig. 1–2 to closed Freyd categories.

To present these results, we will add to the type theory a set τ of *base types*—so that we have a bigger set of types—and two sets $\sigma_{\mathsf{val}}, \sigma_{\mathsf{prod}}$ of *function-symbols.* The function-symbols in σ_{val} represent effect-free functions (i.e. value morphisms); the function-symbols in σ_{prod} represent effectful functions (i.e. producer morphisms). Each function-symbol is equipped with an *arity*—which is a finite sequence of types—and a *result type.* These types can involve the base types. The triple $(\tau, \sigma_{\mathsf{val}}, \sigma_{\mathsf{prod}})$, together with all the arities and result types, is called a *signature.*

From now until the end of the proof of Prop. 5.2, fix a signature $S = (\tau, \sigma_{\mathsf{val}}, \sigma_{\mathsf{prod}})$. The *terms generated by* S are defined as in Fig. 1, with the additional rule

$$\frac{\Gamma \vdash^{\mathsf{v}} V_0 : A_0 \quad \cdots \quad \Gamma \vdash^{\mathsf{v}} V_{m-1} : A_{m-1}}{\Gamma \vdash^{\mathsf{v}} f(V_0, \ldots, V_{m-1}) : B}$$

for each function-symbol $f \in \sigma_{\mathsf{val}}$ whose arity is $(A_0, \ldots, A_{m-1})$ and whose result type is B, and with the additional rule

$$\frac{\Gamma \vdash^{\mathsf{v}} V_0 : A_0 \quad \cdots \quad \Gamma \vdash^{\mathsf{v}} V_{m-1} : A_{m-1}}{\Gamma \vdash^{\mathsf{p}} f(V_0, \ldots, V_{m-1}) : B}$$

for each function-symbol $f \in \sigma_{\mathsf{prod}}$ whose arity is $(A_0, \ldots, A_{m-1})$ and whose result type is B.

We do not allow a term such as $f(M_0, \ldots, M_{m-1})$, where $M_0, \ldots, M_{m-1}$ are producers, because this gives no indication of the order of evaluation of these producers. Rather, we write

$$M_0 \texttt{ to } \mathtt{x}_0.\ \ldots M_{m-1} \texttt{ to } \mathtt{x}_{m-1}.\ f(\mathtt{x}_0, \ldots, \mathtt{x}_{m-1})$$

to indicate that the producers are evaluated left-to-right, for example.

An *interpretation* of the signature S in a Freyd category consists of

- an object $[\![A]\!]$ for each object A—this gives rise to a semantics of types in the obvious way
- a value-morphism $[\![A_0]\!] \times \cdots \times [\![A_{m-1}]\!] \xrightarrow{[\![f]\!]} [\![B]\!]$ for each function-symbol $f \in \sigma_{\mathsf{val}}$ whose arity is $(A_0, \ldots, A_{m-1})$ and whose result type is B
- a producer-morphism $[\![A_0]\!] \times \cdots \times [\![A_{m-1}]\!] \xrightarrow{[\![f]\!]} [\![B]\!]$ for each function-symbol $f \in \sigma_{\mathsf{prod}}$ whose arity is $(A_0, \ldots, A_{m-1})$ and whose result type is B.

It is clear that an interpretation of S in a Freyd category induces a semantics for the terms generated from S.

PROPOSITION 5.1 (Soundness). *For a signature S, and an interpretation for S in a Freyd category, the induced semantics for the terms generated from S validates all the equations of Fig. 1.*

Proof. Straightforward, with a substitution lemma. ∎

A *theory* for the signature S is a congruence $\sim$ on the terms (more accurately, the terms in context) generated by S, respecting substitution (so that related values substituted into related terms give related terms) and respecting weakening (so that, for example, if $\Gamma \vdash^{\mathsf{p}} M \sim M' : B$ then $\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M \sim M' : B$), that includes all the laws of Fig. 1. It is a consequence of Prop. 5.1 that an interpretation of S in a Freyd category induces a theory for S, where two terms are related when they have the same denotation. The converse is also true:

PROPOSITION 5.2 (Completeness). *Given any theory $\sim$ for S, there is a Freyd category, and an interpretation of S in it, such that two terms have the same denotation iff they are related by $\sim$.*

Proof. We construct a Freyd category where

- the objects are the types (involving the base types in τ)
- the value-morphisms from A to B are the equivalence classes w.r.t. $\sim$ of values $\mathtt{x} : A \vdash^{\mathsf{v}} V : B$
- the producer-morphisms from A to B are the equivalence classes w.r.t. $\sim$ of producers $\mathtt{x} : A \vdash^{\mathsf{p}} M : B$

All the structure is easy to define, and well-defined because $\sim$ is a congruence. The interpretation of a base type A is the type A. The interpretation of $f \in \sigma_{\mathsf{val}}$ is the equivalence class of the value

$$\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{m-1} : A_{m-1} \vdash^{\mathsf{v}} f(\mathtt{x}_0, \ldots, \mathtt{x}_{m-1}) : B$$

where f has arity $(A_0, \ldots, A_{m-1})$ and result type B. The interpretation of $f \in \sigma_{\mathsf{prod}}$ is the equivalence class of the producer

$$\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{m-1} : A_{m-1} \vdash^{\mathsf{p}} f(\mathtt{x}_0, \ldots, \mathtt{x}_{m-1}) : B$$

where f has arity $(A_0, \ldots, A_{m-1})$ and result type B. The required equations are easy, if tedious, to verify, as a consequence of the equations in Fig. 1. ∎

To see why we call this a completeness result, introduce the following notation. Let D be a set of equations and E a single equation using the symbols of S. We say that $D \vdash E$ when E can be deduced from D using the equations of Fig. 1 and $D \models E$ when, for every interpretation of S in a Freyd category that validates all the equations of D, the equation E is validated too. Then Prop. 5.1 tells us that $D \vdash E$ implies $D \models E$, while Prop. 5.2 tells us that $D \models E$ implies $D \vdash E$ (take the theory containing precisely those equations deducible from D). In summary, our equational theory provides a sound and complete way of reasoning about Freyd categories.

But ideally we would like to assert not that first-order fragment is a good way of reasoning about Freyd categories, but rather that Freyd categories are a good way of modelling the first-order fragment, which, after all, was our starting point. Prop. 5.2 does not tell us this.

In fact, the relationship between Freyd categories and the first order fragment is closer than we can learn from Prop. 5.1–5.2. For Freyd categories and models of the first-order fragment (defined in a suitably *a priori* way) are equivalent. This is the same as the relationship between cartesian closed categories and simply typed λ-calculus, so we will not discuss it here.

6. κ-CATEGORIES

In previous sections, we defined a fine-grain version of the computational λ-calculus and showed how it can be modelled in a closed Freyd category. Its first order fragment can be modelled in a Freyd category. It is important to distinguish between first order and higher order structure for various purposes, such as in modelling continuations [17], data refinement, and modularity. In this section, we see that Freyd categories are equivalent to a new construct, that of κ-category. It follows that we can model the first order fragment of the λ_{c}-calculus in a κ-category. We shall extend this to modelling the full calculus in a closed κ-category in the next section. The notion of κ-category models environments differently from the way they are modelled in a Freyd category.

We proceed by constructing an indexed category from a Freyd category, then we identify the image of the construction, yielding the notion of κ-category.

DEFINITION 6.1. A *comonoid* in a premonoidal category $\mathcal{K}$ consists of an object $\mathbf{C}$ of $\mathcal{K}$, and central maps $\delta : \mathbf{C} \longrightarrow \mathbf{C} \otimes \mathbf{C}$ and $\nu : \mathbf{C} \longrightarrow I$ making the usual associativity and unit diagrams commute. A *comonoid map* from $\mathbf{C}$ to $\mathbf{D}$ in a premonoidal category $\mathcal{K}$ is a central map $f : \mathbf{C} \longrightarrow \mathbf{D}$ that commutes with the comultiplications and counits of the comonoids.

Given a premonoidal category $\mathcal{K}$, comonoids and comonoid maps in $\mathcal{K}$ form a category $\mathbf{Comon}(\mathcal{K})$ with composition given by that of $\mathcal{K}$. Moreover, any centrality-preserving strict premonoidal functor $H : \mathcal{K} \longrightarrow \mathcal{L}$ lifts to a functor $\mathbf{Comon}(H) : \mathbf{Comon}(\mathcal{K}) \longrightarrow \mathbf{Comon}(\mathcal{L})$. Trivially, any comonoid $\mathbf{C}$ yields a comonad $- \otimes \mathbf{C}$, and any comonoid map $f : \mathbf{C} \longrightarrow \mathbf{D}$ yields a functor from $\mathbf{Kleisli}(- \otimes \mathbf{D})$, the Kleisli category of the comonad $- \otimes \mathbf{D}$, to $\mathbf{Kleisli}(- \otimes \mathbf{C})$, that is the identity on objects. So we have a functor $\mathsf{s}(\mathcal{K}) : \mathbf{Comon}(\mathcal{K})^{\mathrm{op}} \longrightarrow \mathbf{Cat}$. Given a cartesian category $\mathcal{C}$, every object A of $\mathcal{C}$ has a unique comonoid structure, given by the diagonal and the unique map to the terminal object. So $\mathbf{Comon}(\mathcal{C})$ is isomorphic to $\mathcal{C}$. Thus, given a Freyd category $J : \mathcal{C} \longrightarrow \mathcal{K}$, we have a functor $\kappa(J) : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$ given by $\mathsf{s}(\mathcal{K})$ composed with the functor induced by J from $\mathcal{C} \cong \mathbf{Comon}(\mathcal{C})$ to $\mathbf{Comon}(\mathcal{K})$. $\kappa(J)$ is a locally $\mathcal{C}$-indexed category, in the following sense.

Definition 6.2. Let $\mathcal{C}$ be a category.

1. A *strict $\mathcal{C}$-indexed category H* is a functor from $\mathcal{C}^{\mathrm{op}}$ to $\mathbf{Cat}$.
2. A *locally $\mathcal{C}$-indexed category* consists of a set of objects $\mathbf{Ob}\, H$ together with a strict $\mathcal{C}$-indexed category H, where each fibre H_X has the same set of objects $\mathbf{Ob}\, H$ and each reindexing functor H_f is identity on objects.

(Note that if $\mathcal{C}$ is non-empty then it is not necessary to give $\mathbf{Ob}\, H$ explicitly, because it can be recovered as $\mathbf{Ob}\, H_X$ for any object X. In our examples, $\mathcal{C}$ is always non-empty because it has a terminal object.)

Remark 6.3. Our usage of the word "locally" is similar to the usage in "locally small category" or "locally ordered category": the homsets are indexed but the objects are not. A more abstract, but equivalent, definition of locally $\mathcal{C}$-indexed category is as a $[\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$-enriched category.

An important example arises when $\mathcal{C}$ is a cartesian category. We then define the locally $\mathcal{C}$-indexed category $\mathsf{self}\ \mathcal{C}$, in which a morphism from A to B over X is a $\mathcal{C}$-morphism from $X \times A$ to B, and identities, composition and reindexing are given in the evident way.

We would like to characterize the locally indexed categories that arise from the κ construction. A solution is as follows, as we shall see below (Thm. 6.10).

Definition 6.4. A *κ-category* consists of

- a category $\mathcal{C}$ with finite products
- a locally $\mathcal{C}$-indexed category $H : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$ whose class of objects is $\mathbf{Ob}\, \mathcal{C}$
- for every object B, an isomorphism

$$H_{A \times B}(1, \pi^*_{A,B} C) \cong H_A(B, C) \qquad \text{natural in } A \text{ and } C \qquad (1)$$

such that the two functions from $H_1(1, C)$ to $H_{1\times 1}(1, C)$, one given by reindexing and the other given by (1), are equal.

We draw attention to the naturality condition in (1). For this to be meaningful, we need to say how both sides are functorial in A and C, and this is a consequence of the *homset functor* which we shall soon define. Moreover, the condition can be interpreted either

- as two separate naturality conditions—natural in A for any C, and natural in C for any A, or

- as a single naturality condition—natural in the pair ${}_A C$ which ranges not over a product category but over the category $\mathsf{opGroth}\ H$, which we define presently.

Fortunately, as for product categories, the two interpretations can easily be shown equivalent.

We will now define the *opGrothendieck construction* on locally indexed categories; this is important primarily because it enables us to define homset functors—which are of great importance in the theory of functor representations, adjunctions etc.—and secondarily because it allows us to make sense of the "joint naturality" condition that we just discussed.

DEFINITION 6.5. Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. Then $\mathsf{opGroth}\ \mathcal{D}$ is the ordinary category defined as follows:

- an object of $\mathsf{opGroth}\ \mathcal{D}$ is a pair ${}_\Gamma A$ where $\Gamma \in \mathbf{Ob}\,\mathcal{C}$ and $A \in \mathbf{Ob}\,\mathcal{D}$
- a morphism from ${}_\Gamma A$ to ${}_{\Gamma'} B$ in $\mathsf{opGroth}\ \mathcal{D}$ consists of a pair ${}_k f$ where $k : \Gamma' \longrightarrow \Gamma$ in $\mathcal{C}$ and $f : A \longrightarrow B$ in $\mathcal{D}_{\Gamma'}$

with the evident identity and composition.

Thus $\mathcal{D}$ gives us a homset functor from $\mathsf{opGroth}\ (\mathcal{D}^{\mathrm{op}} \times \mathcal{D})$ to $\mathbf{Set}$ taking ${}_\Gamma(X, Y)$ to $\mathcal{D}_\Gamma(X, Y)$.

Remark 6.6. We write an object of $\mathsf{opGroth}\ \mathcal{D}$ as ${}_\Gamma A$ rather than (Γ, A) so that the homset $\mathcal{D}_\Gamma(X, Y)$ can be read as the homset functor $\mathcal{D} : \mathsf{opGroth}\ (\mathcal{D}^{\mathrm{op}} \times \mathcal{D}) \longrightarrow \mathbf{Set}$ applied to the object ${}_\Gamma(X, Y)$.

Using these concepts, we will give a variant of Def. 6.4 which has the advantage that it requires no coherence condition. It is motivated by the notion of "strong adjunction" which appeared in [8], as explained in Appendix B.

DEFINITION 6.7. A *strong κ-category* consists of

- a category $\mathcal{C}$ with finite products
- a locally $\mathcal{C}$-indexed category $H : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$ whose class of objects is $\mathbf{Ob}\,\mathcal{C}$
- a functor $\mathcal{L}$ from $\mathsf{opGroth}\ H$ to $\mathbf{Set}$—we call an element $g \in \mathcal{L}_A B$ an *oblique morphism* over A to B and we write $\xrightarrow[A]{g} B$
- for each object B, an isomorphism

$$H_A(B, C) \cong \mathcal{L}_{A \times B} \pi^*_{A,B} C \qquad \text{natural in } A \text{ and } C \tag{2}$$

The oblique morphisms correspond to the producer-morphisms from A to B in the Freyd category approach; they are the denotations of producers. Specifically, a producer $\Gamma \vdash^{\mathsf{p}} M : B$ denotes an oblique morphism over $[\![\Gamma]\!]$ to $[\![B]\!]$. The morphisms of H, by contrast, are not the denotations of anything, but they help to organize the semantics. Because $\mathcal{L}$ is a functor, it provides "reindexing" and "composition" for oblique morphisms:

- For each oblique morphism $\xrightarrow[A]{g} B$ and $\mathcal{C}$-morphism $A' \xrightarrow{k} A$, we define the *reindexed* oblique morphism $\xrightarrow[A']{k^* g} B$ to be $(\mathcal{L}_k \underline{Y})g$.

- For each oblique morphism $\xrightarrow[A]{g} B$ and $\mathcal{D}$-morphism $B \xrightarrow[A]{h} B'$, we define the *composite* oblique morphism $\xrightarrow[A]{g;h} B'$ to be $(\mathcal{L}_\Gamma h)g$.

These operations satisfy identity, associativity and reindexing laws:

$$
\begin{array}{rcl}
g;\mathsf{id} & = & g \\
g;(h;h') & = & (g;h);h' \\
\mathsf{id}^* g & = & g \\
(k';k)^* g & = & k'^*(k^* g) \\
k^*(g;h) & = & (k^* g);(k^* h)
\end{array}
$$

where g is an oblique morphism. Conversely, these operations and equations give us a functor $\mathcal{L}$ from $\mathsf{opGroth}\ H$ to $\mathbf{Set}$.

The continuation example illustrates well the advantage of strong κ-categories.

- We define the locally $\mathbf{Set}$-indexed category H by saying that a morphism over A from B to C—i.e. an element of $H_A(B,C)$—is a function from $A \times (C \to \mathsf{Ans})$ to $B \to \mathsf{Ans}$. Identities, composition and reindexing in H are defined in the evident way, and all the associativity laws etc. are obvious.
- An oblique morphism over A to B is a function from $A \times (B \to \mathsf{Ans})$ to Ans. This ensures that a producer $\Gamma \vdash^{\mathsf{p}} M : B$ will denote a function from $[\![\Gamma]\!] \times ([\![B]\!] \to \mathsf{Ans})$ to Ans, which is what we want. Again, composition and reindexing are defined in the evident way, and associativity laws etc. are obvious.
- We need for (2) an isomorphism of the form

$$\mathbf{Set}(A \times (C \to \mathsf{Ans}), B \to \mathsf{Ans}) \cong \mathbf{Set}((A \times B) \times (C \to \mathsf{Ans}), \mathsf{Ans}) \tag{3}$$

  It is evident what this isomorphism should be and easy to check the required naturality.

Notice how simple the definition of composition is in the above example and how obvious the associativity law is. This contrasts with the Freyd category in Example 4.3 where currying is required to define composition; in the strong κ-category approach, the currying is all contained in the isomorphism (3). Of course, we could have set up the Freyd category so that a morphism from A to B is a function from $B \to \mathsf{Ans}$ to $A \to \mathsf{Ans}$, but this would not have given semantics of producers in the form we wanted. Just as the Freyd category approach provides more flexibility than the strong monad approach in organizing a model, we see that the strong κ-category approach provides even greater flexibility than either the Freyd category or κ-category approaches. For it allows us to set up the oblique morphisms in the most computationally appropriate way, and set up the morphisms of H so as to make composition simple.

To a lesser extent we can take advantage of this flexibility in organizing the global store model too.

- We define the locally $\mathbf{Set}$-indexed category H by saying that a morphism over A from B to C—i.e. an element of $H_A(B,C)$—is a function from $A \times (S \times B)$ to $S \times C$. Identities, composition and reindexing in H are defined in the evident way, and all the associativity laws etc. are obvious.

- An oblique morphism over A to B is a function from $S \times A$ to $S \times B$. This ensures that a producer $\Gamma \vdash^{\mathsf{p}} M : B$ will denote a function from $S \times [\![\Gamma]\!]$ to $S \times [\![B]\!]$, which is what we want. Again, composition and reindexing are defined in the evident way, and associativity laws etc. are obvious.

- We need for (2) an isomorphism of the form

$$\mathbf{Set}(A \times (S \times B), S \times C) \cong \mathbf{Set}(S \times (A \times B), S \times C)$$

  It is evident what this isomorphism should be and easy to check the required naturality.

PROPOSITION 6.8. *Given a strong κ-category and an object B, there is a unique element of $\mathcal{L}_B B$, which we call prod_B, such that the isomorphism (2) takes a morphism $h : B \longrightarrow C$ in H_A to $(\pi'^*_{A,B}\mathsf{prod}_B); (\pi^*_{A,B} h)$.*

Proof. This is an instance of the locally indexed version of the Yoneda Lemma, and can be proved directly in the same style as the Yoneda Lemma. prod_B is obtained by applying isomorphism (2) to the identity on B over 1, and then reindexing along $A \cong 1 \times A$. ∎

PROPOSITION 6.9. *Given a κ-category $H : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$, there is an indexed functor $\mathsf{inc} : \mathsf{s}(\mathcal{C}) \longrightarrow H$ as follows: for each A in $\mathcal{C}$, we have a functor from $\mathsf{s}(\mathcal{C}_A)$ to H_A. On objects, it is the identity. To define inc_1 on arrows, given $f : A \longrightarrow B$ in $\mathcal{C}$, consider the arrow $\iota_B : 1 \longrightarrow B$ in H_B corresponding to id_B in H_1. Applying H_f to it gives a map $H_f(\iota_B) : 1 \longrightarrow B$ in H_A, or equivalently, under the adjunction, a map from A to B in H_1. Define $\mathsf{inc}_1(f)$ to be that map.*

$$\begin{tikzcd}
\mathbf{1} & A \arrow[r, "f"] & B & \mathbf{1} \\
A \arrow[d, "\mathsf{inc}_1(f)"'] & \mathbf{1} \arrow[d, "H_f(\iota_B)"'] & \mathbf{1} \arrow[d, "\iota_B"] \arrow[l, phantom, "\xleftarrow{H_f}"] & B \arrow[d, "\mathsf{id}_B"] \\
B & B & B & B
\end{tikzcd}$$

This plus naturality determines the rest of the structure.

Proof. It is immediate that inc_1 preserves identities, and one can prove that it preserves composition: this follows by proving that for any map $f : A \longrightarrow B$ in $\mathcal{C}$ and any map $g : 1 \longrightarrow C$ in H_B, the map $H_f(g)$ corresponds to the composite in H_1 of $\mathsf{inc}_1(f)$ with the adjoint correspondent to g. Moreover, this yields a functor inc_A for every A, with naturality as required. ∎

Using Prop. 6.9, we can exhibit the relationship between Freyd categories and κ-categories. This forms the basis for the first main result of the paper, Theorem 6.10.

THEOREM 6.10. *Given a cartesian category $\mathcal{C}$, the following are equivalent (more precisely, the 2-categories defined as in Appendix C are 2-equivalent):*

1. *to give a Freyd category $J : \mathcal{C} \longrightarrow \mathcal{K}$*

2. *to give a strong κ-category $(H, \mathcal{L})$ with base $\mathcal{C}$*

3. to give a κ-category $H : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$.

The equivalence of (2) and (3) is straightforward. For the equivalence of (3) and (1), we observe from the definition of κ-category that for each projection $\pi : B \times A \longrightarrow B$ in $\mathcal{C}$, the functor $H_\pi : H_B \longrightarrow H_{B\times A}$ has a left adjoint L given on objects by $A \times -$. We denote the isomorphism associated with these adjunctions by

$$\kappa : H_{B\times A}(C, \pi^*_{B,A}C') \cong H_B(C \times A, C').$$

First, for the construction of a κ-category from a Freyd category, we have

PROPOSITION 6.11. *Given a Freyd category $J : \mathcal{C} \longrightarrow \mathcal{K}$, the functor given by $\kappa(J) : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$ is a κ-category.*

Proof. It follows immediately from the construction of $\kappa(J)$ in Section 6 that for each object A of $\mathcal{C}$, we have $\mathbf{Ob}\,\kappa(J)_A = \mathbf{Ob}\,\mathcal{C}$, and that for each arrow $f : A \longrightarrow B$ in $\mathcal{C}$, the functor $\kappa(J)_f$ is the identity on objects. Moreover, the existence of the partial adjoints to each $\kappa(J)_\pi$ follows directly from the construction and the fact that $\mathcal{C}$ is symmetric. Naturality and the coherence condition also follows directly from the construction. ∎

Now for the converse:

PROPOSITION 6.12. *Let $\mathcal{C}$ be a cartesian category. Given a κ-category $H : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$, there is a Freyd category $J : \mathcal{C} \longrightarrow \mathcal{K}$, unique up to isomorphism, for which H is isomorphic to $\kappa(J)$.*

Proof. Define $\mathcal{K}$ to be H_1. For each object A of $\mathcal{K}$, equally A an object of $\mathcal{C}$ since $\mathbf{Ob}\,H_1 = \mathbf{Ob}\,\mathcal{C}$ (as is immediate from the first clause of the definition applied to the case $A = 1$), define $- \times A : \mathcal{K} \longrightarrow \mathcal{K}$ by the composite $L \circ H_!$ where $! : A \longrightarrow 1$ is the unique map in $\mathcal{C}$ from A to 1. Note that ! is of the form π, so the left adjoint exists. Moreover, for each map $g : C \longrightarrow C'$ in $\mathcal{K}$, we have $g \times A : C \times A \longrightarrow C' \times A$. The rest of the data and axioms to make $\mathcal{K}$ a symmetric premonoidal category arise by routine calculation, using the symmetric monoidal structure of $\mathcal{C}$ determined by its finite product structure, and by the naturality condition.

Define $J : \mathcal{C} \longrightarrow \mathcal{K}$ by inc_1 as in Prop. 6.9. It follows from naturality that for a map $f : A \longrightarrow B$ in $\mathcal{C}$, and for a map $g : C \longrightarrow D$ in H_B, we have that $H_f(g)$ is given by the composite of $J(id_C \times f)$ with the adjoint correspondent of g. Naturality further implies that $(\mathsf{inc}_1 -) \times A$ agrees with $\mathsf{inc}_1(- \times A)$. It follows from functoriality of the H_f's that every map in $\mathcal{C}$ is sent into the centre of $\mathcal{K}$. Functoriality plus naturality similarly imply that all the structural maps are preserved. So J is an identity on objects strict symmetric premonoidal functor.

It follows directly from our construction of J that $\kappa(J)$ is isomorphic to H. Moreover, $J : \mathcal{C} \longrightarrow \mathcal{K}$ is fully determined by H since $\mathcal{C}$ is fixed, $\mathcal{K}$ must be H_1 up to isomorphism, with premonoidal structure as given, and J must agree on maps with the construction as we have given it. Hence, J is unique up to isomorphism.

The final line of the theorem follows routinely. ∎

Now we can see how one models environments in a κ-category: a context is modelled by an object of the base category $\mathcal{C}$, with the finite products of $\mathcal{C}$ modelling concatenation of contexts. A term of type τ in context Γ is modelled by an arrow in the fibre over $[\![\Gamma]\!]$ from $\mathbf{1}$ to $[\![\tau]\!]$. Substitution of a value for a variable is modelled by the functoriality of a κ-category with respect to maps in $\mathcal{C}$. So the emphasis here is upon substitution of a value for a variable as a primitive operation.

7. CLOSED FREYD-CATEGORIES, CLOSED κ-CATEGORIES, AND λ_c-MODELS

In previous sections, we have considered two ways of modelling the first order fragment of the λ_c-calculus. In this section, we extend that to model higher-order structure, allowing us two ways to model the λ_c-calculus [9]. We define and relate closed Freyd categories and closed κ-categories with λ_c-models.

DEFINITION 7.1. A *closed strong κ-category* consists of a strong κ-category $(\mathcal{C}, H, \mathcal{L}, \ldots)$ together with, for each pair of objects A, B, a representation for the functor $\lambda X.\mathcal{L}_{X\times A}B$.

DEFINITION 7.2. A *closed κ-category* is a κ-category $H : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$ together with, for each pair of objects A, B, a representation for the functor $\lambda X.H_X(A, B)$.

The main result of this section:

THEOREM 7.3. *Given a cartesian category $\mathcal{C}$, the following are equivalent (more precisely, the 2-categories defined as in Appendix C are 2-equivalent):*

1. *to give a closed Freyd category $J : \mathcal{C} \longrightarrow \mathcal{K}$*
2. *to give a closed strong κ-category $(H, \mathcal{L})$ with base $\mathcal{C}$*
3. *to give a closed κ-category $H : \mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Cat}$*
4. *to give a strong monad on $\mathcal{C}$, with Kleisli exponentials.*

The equivalence of (1) and (4) is Thm 4.7. The equivalence of (1)–(3) follows from Thm. 6.10: the functors whose representation is required (for given objects A,B) are isomorphic.

THEOREM 7.4. *Closed Freyd categories, closed κ-categories and closed strong κ-categories form sound and complete classes of models for the computational λ-calculus.*

We have already seen this for closed Freyd categories in Sect. 5. The result for closed κ-categories follows from Thm. 7.3.

8. CONCLUSIONS

We have examined various categorical models of call-by-value programming with effects, using strong monads, Freyd categories, and strong κ-categories. (We also used κ-categories, but, as we mentioned in Sect. 1, this was just as a stepping-stone towards strong κ-categories.) We have seen that these various categorical models are equivalent (and also mentioned in Sect. 5 that they are equivalent to models of the fine-grain CBV equational theory, defined in a suitably a priori way). But equivalence does not means that the various models correspond exactly, only that they correspond up to isomorphism (identity-on-objects isomorphism preserving all structure on the nose), and there are significant differences contained within these isomorphisms:

- A λ_{c}-model (cartesian category with strong monad and Kleisli exponentials) corresponds to a closed Freyd category where the isomorphism

$$\mathcal{K}(X \times A, B) \cong \mathcal{C}(X, A \to B)$$

  is an identity. Thus the structure of a closed Freyd category provides more flexibility than the structure of a λ_{c}-model, as we explained in Sect. 4—we can separately decide how to model values and how to model producers.

- A Freyd category corresponds to a strong κ-category where the isomorphism

$$H_A(B, C) \cong \mathcal{L}_{A \times B} C$$

  is an identity. Thus the structure of a strong κ-category provides more flexibility than the structure of a Freyd category, as we explained in Sect. 6—we can separately decide how to interpret effects (in $\mathcal{L}$) and how to organize the environment housekeeping (in H).

In summary, the strong κ-category approach provides the most flexibility and the strong monad approach provides the least. But while flexibility is an advantage when constructing particular models for call-by-value, the rigidity of the strong monad approach is useful for proving results about all models of CBV, because strong monads are very easy to reason about. And the Freyd category approach has its advantages too: because it is so close to the syntax, it gives us a useful way of thinking about the term model.

Acknowledgements

Thanks to Carsten Führmann, Peter O'Hearn and anonymous referees for helpful discussions and comments.

REFERENCES

[1] R. Blackwell, G. M. Kelly, and A. J. Power. Two-dimensional monad theory. *J. Pure Appl. Algebra*, 59:1–41, 1989.

[2] R. Blute, J. R. B. Cockett, and R. A. G. Seely. Categories for computation in context and unified logic. *Journal of Pure and Applied Algebra*, 116:49–98, 1997.

[3] M. Hasegawa. Decomposing typed lambda calculus into a couple of categorical programming languages. In D. Pitt, D. E. Rydeheard, and P. T. Johnstone, editors, *Proceedings of the 6th International Conference on Category Theory and Computer Science (CTCS'95)*, volume 953 of *LNCS*, pages 200–219. Springer, August 1995.

[4] B. Jacobs. *Categorical Type Theory*. PhD thesis, University of Nijmegen, 1991.

[5] G.M. Kelly. *The basic concepts of enriched categories*. CUP (1982).

[6] G. M. Kelly and A. J. Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *J. Pure Appl. Algebra*, 89:163–179, 1993.

[7] P. B. Levy. Call-by-push-value: a subsuming paradigm (extended abstract). In J.-Y Girard, editor, *Typed Lambda-Calculi and Applications*, volume 1581 of *LNCS*, pages 228–242, L'Aquila, 1999. Springer.

[8] P. B. Levy. *Call-by-push-value.* PhD thesis, Queen Mary, University of London, 2001.

[9] E. Moggi. Computational Lambda calculus and Monads, *Proc. LICS 89*, IEEE Press (1989) 14-23.

[10] E. Moggi, Notions of computation and monads, Information and Computation 93 (1991) 55-92.

[11] A. J. Power. Premonoidal categories as categories with algebraic structure. *Theoretical Computer Science*, 278(1–2):303–321, 2002.

[12] A. J. Power and E. P. Robinson. Premonoidal categories and notions of computation. *Math. Struct. in Comp. Sci.*, 7(5):453–468, October 1997.

[13] A. J. Power and H. Thielecke. Environments, continuation semantics and indexed categories. In *Proceedings TACS'97*, volume 1281 of *LNCS*, pages 391–414. Springer-Verlag, Berlin, 1997.

[14] A. J. Power and H. Thielecke. Closed Freyd- and kappa-categories. In *Proc. ICALP '99*, volume 1644 of *LNCS*, pages 625–634. Springer-Verlag, Berlin, 1999.

[15] E. P. Robinson. Variations on algebra: monadicity and generalisations of algebraic theories. *Math. Struct. in Comp. Sci.* to appear.

[16] E.P. Robinson and G. Rosolini, Categories of partial maps, Information and Computation 79 (1988) 95-130.

[17] H. Thielecke. *Categorical Structure of Continuation Passing Style.* PhD thesis, University of Edinburgh, 1997. Also available as technical report ECS-LFCS-97-376.

APPENDIX A: PREMONOIDAL CATEGORIES

We recall the definitions of premonoidal category and strict premonoidal functor, and symmetries for them, as introduced in [12] and further studied in [11]. A premonoidal category is a generalisation of the concept of monoidal category: it is essentially a monoidal category except that the tensor need only be a functor of two variables and not necessarily be bifunctorial, i.e., given maps $f : A \longrightarrow B$ and $f' : A' \longrightarrow B'$, the evident two maps from $A \otimes A'$ to $B \otimes B'$ may differ.

Historically, for instance for the simply typed λ-calculus, environments have been modelled by finite products. More recently (within the past decade or so), monoidal structure has sometimes been used, for instance when one wants to incorporate an account of partiality [16]. In the presence of stronger computational effects, an even weaker notion is required. If the computational effects are strong enough for the order of evaluation of $f : A \longrightarrow B$ and $f' : A' \longrightarrow B'$ to be observable, as for instance in the case of continuations [17], then the monoidal laws cannot be satisfied. The leading examples for us of such stronger computational effects are those given by continuations. However, for a simple example of a premonoidal category that may be used for a crude account of state [12], consider the following.

EXAMPLE A.1. Given a symmetric monoidal category $\mathcal{C}$ together with a specified object S, define the category $\mathcal{K}$ to have the same objects as $\mathcal{C}$, with $\mathcal{K}(A,B) = \mathcal{C}(S \otimes A, S \otimes B)$, and with composition in $\mathcal{K}$ determined by that of $\mathcal{C}$. For any object A of $\mathcal{C}$, one has functors $A \otimes - : \mathcal{K} \longrightarrow \mathcal{K}$ and $- \otimes A : \mathcal{K} \longrightarrow \mathcal{K}$, but they do not satisfy the bifunctoriality condition above, hence do not yield a monoidal structure on $\mathcal{K}$. They do yield a premonoidal structure, as we define below.

In order to make precise the notion of a premonoidal category, we need some auxiliary definitions.

DEFINITION A.2. A *binoidal category* is a category $\mathcal{K}$ together with, for each object A of $\mathcal{K}$, functors $h_A : \mathcal{K} \longrightarrow \mathcal{K}$ and $k_A : \mathcal{K} \longrightarrow \mathcal{K}$ such that for each pair (A,B) of objects of $\mathcal{K}$, $h_A B = k_B A$. The joint value is denoted $A \otimes B$.

DEFINITION A.3. An arrow $f : A \longrightarrow A'$ in a binoidal category is *central* if for every arrow $g : B \longrightarrow B'$, the following diagrams commute:

$$\begin{tikzcd} A \otimes B \arrow[r, "A\otimes g"] \arrow[d, "f\otimes B"'] & A \otimes B' \arrow[d, "f\otimes B'"] \\ A' \otimes B \arrow[r, "A'\otimes g"] & A' \otimes B' \end{tikzcd} \qquad \begin{tikzcd} B \otimes A \arrow[r, "g\otimes A"] \arrow[d, "B\otimes f"'] & B' \otimes A \arrow[d, "B'\otimes f"] \\ B \otimes A' \arrow[r, "g\otimes A'"] & B' \otimes A' \end{tikzcd}$$

Moreover, given a binoidal category $\mathcal{K}$, a natural transformation $\alpha : g \Longrightarrow h : \mathcal{B} \longrightarrow \mathcal{K}$ is called *central* if every component of α is central.

DEFINITION A.4. A *premonoidal category* is a binoidal category $\mathcal{K}$ together with an object I of $\mathcal{K}$, and central natural isomorphisms a with components $(A \otimes B) \otimes C \longrightarrow A \otimes (B \otimes C)$, l with components $A \longrightarrow A \otimes I$, and r with components $A \longrightarrow I \otimes A$, subject to two equations: the pentagon expressing coherence of a, and the triangle expressing coherence of l and r with respect to a (see [5] for an explicit depiction of the diagrams).

Now we have the definition of a premonoidal category, it is routine to verify that Example A.1 is an example of one. There is a general construction that yields premonoidal categories too:

PROPOSITION A.5. *Given a strong monad T on a symmetric monoidal category $\mathcal{C}$, the Kleisli category* **Kleisli**(T) *for T is always a premonoidal category, with the functor $J : \mathcal{C} \longrightarrow$* **Kleisli**$(T)$ *preserving premonoidal structure strictly: of course, a monoidal category such as $\mathcal{C}$ is trivially a premonoidal category.*

So a good source of examples of premonoidal categories is provided by Eugenio Moggi's work on monads as notions of computation [10], and indeed there is a representation result as explained in [12].

DEFINITION A.6. Given a premonoidal category $\mathcal{K}$, define the *centre* of $\mathcal{K}$, denoted $\mathsf{Z}(\mathcal{K})$, to be the subcategory of $\mathcal{K}$ consisting of all the objects of $\mathcal{K}$ and the central morphisms.

For an example of the centre of a premonoidal category, consider Example A.1 for the case of $\mathcal{C}$ being the category **Set** of small sets, with symmetric monoidal structure given by finite products. Suppose S has at least two elements. Then the centre of $\mathcal{K}$ is precisely **Set**. In general, given a strong monad on a symmetric monoidal category, the base category $\mathcal{C}$ need not be the centre of $\mathbf{Kleisli}(T)$. But, modulo the condition that $J : \mathcal{C} \longrightarrow \mathbf{Kleisli}(T)$ be faithful, or equivalently, the mono requirement [10, 12], i.e., the condition that the unit of the adjunction be pointwise monomorphic, it must be a subcategory of the centre.

The functors h_A and k_A preserve central maps. So we have

PROPOSITION A.7. *The centre of a premonoidal category is a monoidal category.*

This proposition allows us to prove a coherence result for premonoidal categories, directly generalising the usual coherence result for monoidal categories. Details appear in [12].

DEFINITION A.8. A *symmetry* for a premonoidal category is a central natural isomorphism with components $c : A \otimes B \longrightarrow B \otimes A$, satisfying the two conditions $c^2 = 1$ and equality of the evident two maps from $(A \otimes B) \otimes C$ to $C \otimes (A \otimes B)$. A *symmetric* premonoidal category is a premonoidal category together with a symmetry.

All of the examples of premonoidal categories we have discussed so far are symmetric, and in fact, symmetric premonoidal categories are those of primary interest to us, and seem to be those of primary interest in denotational semantics in general. For an example of a premonoidal category that is not symmetric, consider, given any category $\mathcal{C}$, the category $End_u(\mathcal{C})$ whose objects are functors from $\mathcal{C}$ to itself, and for which an arrow from h to k is a $\mathcal{C}$-indexed family of arrows $\alpha(A) : h(A) \longrightarrow k(A)$ in $\mathcal{C}$, i.e., what would be a natural transformation from h to k but without assuming commutativity of the naturality squares. Then, this category, together with the usual composition of functors, has the structure of a strict premonoidal category, i.e., a premonoidal category in which all the structural isomorphisms are identities, which is certainly not symmetric.

APPENDIX B: STRONG ADJUNCTIONS

To explain where the strong κ-category definition (Def. 6.7) came from, we look at the following definition which appeared (and was carefully motivated with respect to call-by-push-value) in [8].

DEFINITION B.1. A *strong adjunction* from a cartesian category $\mathcal{C}$ to a locally $\mathcal{C}$-indexed category $\mathcal{D}$ consists of

- a functor $\mathcal{O}$ from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**
- for each $B \in \mathbf{Ob}\,\mathcal{C}$, an object $FB \in \mathbf{Ob}\,\mathcal{D}$ together with an isomorphism

$$\mathcal{D}_A(FB, C) \cong \mathcal{O}_{A \times B} C \pi^*_{A,B} C \quad \text{natural in } A \text{ and } C \tag{4}$$

- for each $B \in \mathbf{Ob}\,\mathcal{D}$, an object $UB \in \mathbf{Ob}\,\mathcal{C}$ together with an isomorphism

$$\mathcal{C}(A, UB) \cong \mathcal{O}_A B \quad \text{natural in } A \tag{5}$$

It is shown in [8] that giving a strong adjunction from $\mathcal{C}$ to $\mathcal{D}$ is equivalent to giving an adjunction (in the usual sense) from $\mathsf{self}\ \mathcal{C}$ (defined in Sect. 6) to $\mathcal{D}$. Thus it gives us a monad on $\mathsf{self}\ \mathcal{C}$ i.e. a strong monad on $\mathcal{C}$. This is the reason for using the word "strong".

As with ordinary adjunctions, we can say that a strong adjunction from $\mathcal{C}$ to H is *Kleisli* when H has the same objects as $\mathcal{C}$ and $FB = B$ for every object $B \in \mathbf{Ob}\,\mathcal{C}$. In this situation it is customary to write TB rather than UB, and $\mathcal{L}$ rather than $\mathcal{O}$ so that the two isomorphisms look like this:

$$H_A(B, C) \quad \mathcal{L}_{A\times B}\pi^*_{A,B}C \cong \text{ natural in } A \text{ and } C \tag{6}$$

$$\mathcal{C}(A, TB) \cong \mathcal{L}_A B \quad \text{natural in } A \tag{7}$$

For the sake of modelling the first-order fragment of fine-grain CBV, we do not require (7) and so we are led to Def. 6.7.

APPENDIX C: DEFINING THE 2-CATEGORIES

C.1. Aim

In the paper we look at the following 8 categorical structures

- cartesian category
- λ_{c}-model
- Freyd category
- closed Freyd category
- κ-category
- closed κ-category
- strong κ-category
- closed strong κ-category

For each of these structures, we can define a notion of functor and natural isomorphism and so we obtain a 2-category (actually a **Grpd**-enriched category, because the 2-cells are isomorphisms). The aim of this appendix is to define all these 2 categories. We will give the definition explicitly for λ_{c}-models and strong κ-categories; the other 6 definitions are entirely analogous.

Each of our structures includes a "value category" $\mathcal{C}$, and its groupoid of isomorphisms **Isos** $\mathcal{C}$ plays a key role in our definitions. It is up to isomorphism in $\mathcal{C}$ that functors preserve object-operations, and everything is functorial or natural in **Isos** $\mathcal{C}$. However, we do not require this functoriality/naturality from the outset, but deduce it from other assumptions, so as to make clear that all our definitions are purely equational.

We have not given a convincing explanation of why isomorphisms in $\mathcal{C}$ should be so important—after all, $\mathcal{C}$ is just one part of the structure. Further research is certainly needed on these issues.

C.2. Algebraic Structure

There is another, well established approach to forming 2-categories, which we cannot use because it does not work for some of our examples. It proceeds as follows. We first express the objects as algebras for a monad on an already known 2-category such as **Cat** or $[\rightarrow, \mathbf{Set}] - \mathbf{Cat}$. We then use the definition given in [1] of morphism and 2-cell between such algebras, inherited from the morphisms and 2-cells in the base category. Such a monad is usually easy to describe, because, as shown in [6], if it has a rank then it must be given by an *algebraic structure*, meaning a pair (S, E) where S is a kind of "signature" and E a kind of "set of equations". A most helpful explanation of this material is given in [15].

Out of our 8 structures, this algebraic structure approach works for cartesian categories (given by algebraic structure on **Cat**), for Freyd categories (on $[\rightarrow, \mathbf{Set}] - \mathbf{Cat}$ [11]) and for closed Freyd categories (on $[\rightarrow, \mathbf{Set}] - \mathbf{Cat}$ enriched over **Grpd**). However, it does not work for λ_{c}-models, because of the Kleisli exponentials, nor for the various κ-categories.

C.3. λ_{c}-models

LEMMA C.1. *In a λ_c model $(\mathcal{C}, T, \ldots)$ all the primitive operations on homsets*

$$1 \xrightarrow{\mathsf{id}_A} \mathcal{C}(A, A)$$

$$\mathcal{C}(A, B) \times \mathcal{C}(B, C) \xrightarrow{;_{ABC}} \mathcal{C}(A, C)$$

$$1 \xrightarrow[\cong]{()_A} \mathcal{C}(A, 1)$$

$$\mathcal{C}(A, B) \times \mathcal{C}(A, C) \xrightarrow[\cong]{(,)_{ABC}} \mathcal{C}(A, B \times C)$$

$$\mathcal{C}(A, B) \xrightarrow{T_{AB}} \mathcal{C}(TA, TB)$$

$$1 \xrightarrow{\eta_A} \mathcal{C}(A, TA)$$

$$1 \xrightarrow{\mu_A} \mathcal{C}(T^2 A, TA)$$

$$1 \xrightarrow{t_{AB}} \mathcal{C}(A \times TB, T(A \times B))$$

$$\mathcal{C}(A \times B, TC) \xrightarrow[\cong]{\mathsf{curry}} \mathcal{C}(A, B \rightarrow_{\mathsf{Kl}} C)$$

are natural as A, B, C range over **Isos** $\mathcal{C}$, *when we regard*

- $\mathcal{C}(-,-)$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Set**
- 1 *as a functor from* 1 *to* **Isos** $\mathcal{C}$
- $\times$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Isos** $\mathcal{C}$
- T *as a functor from* **Isos** $\mathcal{C}$ *to* **Isos** $\mathcal{C}$
- $\to_{\mathsf{Kl}}$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Isos** $\mathcal{C}$

in the evident way.

DEFINITION C.2. A λ_{c} *functor* from a λ_{c} model $(\mathcal{C}, T, \ldots)$ to another $(\mathcal{C}', T' \ldots)$ consists of

- a function $\mathbf{Ob}\, F$ from $\mathbf{Ob}\, \mathcal{C}$ to $\mathbf{Ob}\, \mathcal{C}'$
- functions

$$\mathcal{C}(A,B) \xrightarrow{F(A,B)} \mathcal{C}'(FA,FB) \qquad \text{for each } A, B \in \mathbf{Ob}\, \mathcal{C}$$

- "coherence" morphisms in **Isos** $\mathcal{C}'$, up to which F preserves each object operation

$$\mathbf{1} \qquad F1 \xrightarrow{F^1} 1'$$

$$\times \qquad F(A \times B) \xrightarrow{F^{\times}_{AB}} FA \times' FB \qquad \text{for each } A, B \in \mathbf{Ob}\, \mathcal{C}$$

$$\boldsymbol{T} \qquad FTA \xrightarrow{F^{T}_{A}} T'FA \qquad \text{for each } A \in \mathbf{Ob}\, \mathcal{C}$$

$$\to_{\mathsf{Kl}} \qquad F(A \to_{\mathsf{Kl}} B) \xrightarrow{F^{\to}_{AB}} FA \to'_{\mathsf{Kl}} FB \qquad \text{for each } A, B \in \mathbf{Ob}\, \mathcal{C}$$

- such that F preserves, up to the coherence isomorphisms, each of the primitive operations listed in Lemma C.1. For example:

  preservation of ;

$$\begin{tikzcd}
\mathcal{C}(A,B) \times \mathcal{C}(B,C) \arrow[r, "F(A,B) \times F(B,C)"] \arrow[d, ";_{ABC}"'] & \mathcal{C}'(FA,FB) \times \mathcal{C}'(FB,FC) \arrow[d, ";'_{(FA)(FB)(FC)}"] \\
\mathcal{C}(A,C) \arrow[r, "F(A,C)"'] & \mathcal{C}'(FA,FC)
\end{tikzcd}$$

  preservation of $(,)$

$$\begin{tikzcd}
\mathcal{C}(A,B) \times \mathcal{C}(A,C) \arrow[rr, "F(A,B) \times F(A,C)"] \arrow[d, "(,)_{ABC}"', "\cong"] & & \mathcal{C}(FA,FB) \times \mathcal{C}(FA,FC) \arrow[d, "\cong"', "(,)'_{(FA)(FB)(FC)}"] \\
\mathcal{C}(A,B \times C) \arrow[r, "F(A,B \times C)"'] & \mathcal{C}'(FA,F(B \times C)) \arrow[r, "\mathcal{C}'(FA,F^{\times}_{BC})"'] & \mathcal{C}'(FA,FB \times FC)
\end{tikzcd}$$

preservation of μ

$$\begin{tikzcd}
1 \arrow[rrrr, "\mathsf{id}"] \arrow[d, "\mu_A"'] & & & & 1 \arrow[d, "\mu'_{FA}"] \\
\mathcal{C}(T^2A, TA) \arrow[rr, "F(T^2A,TA)"'] & & \mathcal{C}'(FT^2A, FTA) \arrow[rr, "\mathcal{C}'(F^{T^2}A, F^TA)"'] & & \mathcal{C}'(T'^2FA, T'FA)
\end{tikzcd}$$

where $F^{T^2}A$ is the composite $FT^2A \xrightarrow{F^TTA} T'FTA \xrightarrow{T'F^TA} T'^2A$

preservation of curry

$$\begin{tikzcd}
\mathcal{C}(A \times B, TC) \arrow[r, "F(A\times B,TC)"] \arrow[d, "\mathsf{curry}_{ABC}"'] & \mathcal{C}'(F(A \times B), FTC) \arrow[r, "\mathcal{C}'(F^{\times}_{AB}, F^T_C)"] & \mathcal{C}'(FA \times' FB, T'FC) \arrow[d, "\mathsf{curry}'_{(FA)(FB)(FC)}"] \\
\mathcal{C}(A, B \to_{\mathsf{Kl}} C) \arrow[r, "F(A,B\to_{\mathsf{Kl}}C)"'] & \mathcal{C}'(FA, F(B \to_{\mathsf{Kl}} C)) \arrow[r, "\mathcal{C}'(FA, F^{\to}_{BC})"'] & \mathcal{C}'(FA, FB \to'_{\mathsf{Kl}} FC)
\end{tikzcd}$$

Remark C.3. The requirement that F preserve identity and composition give an extension of $\mathbf{Ob}\,F$ to a functor from **Isos** $\mathcal{C}$ to **Isos** $\mathcal{C}$, and this functor is used in formulating the remaining conditions, such as the preservation of pairing.

Remark C.4. The structure preservation requirements imply that the homset operation $F(A, B)$ and the coherence isomorphisms are all natural as A, B range over **Isos** $\mathcal{C}$. This is important for defining composition of λ_{c}-functors.

Similar remarks are applicable after Def. C.8.

Definition C.5. Let F, G be λ_{c} functors from $(\mathcal{C}, T, \ldots)$ to $(\mathcal{C}', T' \ldots)$. A λ_{c} *natural isomorphism* from F to G is

- a family of morphisms in **Isos** $\mathcal{C}'$

$$FA \xrightarrow{\alpha A} GA$$

- preserving the homset operations i.e.

$$\begin{tikzcd}
\mathcal{C}(A, B) \arrow[d, "F(A,B)"'] \arrow[dr, "G(A,B)"] & \\
\mathcal{C}'(FA, FB) \arrow[r, "\mathcal{C}'(\alpha A,\alpha B)"'] & \mathcal{C}'(GA, GB)
\end{tikzcd}$$

- and preserving the coherence isomorphisms i.e.

$$\begin{tikzcd} F1 \arrow[r, "F^1"] \arrow[d, "\alpha 1"'] & 1' \arrow[d, "\mathsf{id}"] \\ G1 \arrow[r, "G^1"] & 1' \end{tikzcd} \qquad \begin{tikzcd} F(B \times C) \arrow[r, "F^{\times}_{BC}"] \arrow[d, "\alpha(B \times C)"'] & FB \times' FC \arrow[d, "\alpha B \times \alpha C"] \\ G(B \times C) \arrow[r, "G^{\times}_{BC}"'] & GB \times' GC \end{tikzcd}$$

$$\begin{tikzcd} FTA \arrow[r, "F^T_A"] \arrow[d, "\alpha TA"'] & T'FA \arrow[d, "T'\alpha A"'] \\ GTA \arrow[r, "G^T_A"'] & T'GA \end{tikzcd} \qquad \begin{tikzcd} F(B \to_{\mathsf{Kl}} C) \arrow[r, "F^{\to}_{BC}"] \arrow[d, "\alpha(B \to_{\mathsf{Kl}} C)"'] & FB \to'_{\mathsf{Kl}} FC \arrow[d, "\alpha B \to'_{\mathsf{Kl}} \alpha C"] \\ G(B \to_{\mathsf{Kl}} C) \arrow[r, "G^{\to}_{BC}"'] & GB \to'_{\mathsf{Kl}} GC \end{tikzcd}$$

Remark C.6. The homset operation preservation condition implies that αA is natural as A ranges over **Isos** $\mathcal{C}$. This is important for defining composition of λ_{c} natural transformations.

A similar remark is applicable after Def. C.9.

C.4. Closed Strong κ-Categories

LEMMA C.7. *In a closed strong κ-category $(\mathcal{C}, \mathcal{K}, \iota, \ldots)$ all the primitive operations on homsets*

$$1 \xrightarrow{\mathsf{id}_A} \mathcal{C}(A, A)$$

$$\mathcal{C}(A, B) \times \mathcal{C}(B, C) \xrightarrow{;_{ABC}} \mathcal{C}(A, C)$$

$$1 \xrightarrow[\cong]{()_A} \mathcal{C}(A, 1)$$

$$\mathcal{C}(A, B) \times \mathcal{C}(A, C) \xrightarrow[\cong]{(,)_{ABC}} \mathcal{C}(A, B \times C)$$

$$1 \xrightarrow{\mathsf{id}_{AB}} H_A(B, B)$$

$$H_A(B, C) \times H_A(C, D) \xrightarrow{;_{ABCD}} H_A(B, D)$$

$$\mathcal{C}(A, B) \times H_B(C, D) \xrightarrow{*_{ABCD}} H_A(C, D)$$

$$\mathcal{C}(A, B) \times \mathcal{L}_B C \xrightarrow{*_{BCD}} \mathcal{L}_A C$$

$$\mathcal{L}_B C \times H_B(C, D) \xrightarrow{;_{BDC}} \mathcal{L}_B D$$

$$\mathcal{L}_{A \times B} C \xrightarrow[\cong]{\mathsf{str}_{ABC}} H_A(B, C)$$

$$\mathcal{L}_{A \times B} C \xrightarrow[\cong]{\mathsf{curry}_{ABC}} \mathcal{C}(A, B \to C)$$

are natural as A, B, C range over **Isos** $\mathcal{C}$*, when we regard*

- $\mathcal{C}(-, -)$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Set**
- $\mathcal{L}_- -$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Set**
- $H_-(-, -)$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Set**
- 1 *as a functor from* 1 *to* **Isos** $\mathcal{C}$
- $\times$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Isos** $\mathcal{C}$

- $\to$ *as a functor from* **Isos** $\mathcal{C} \times$ **Isos** $\mathcal{C}$ *to* **Isos** $\mathcal{C}$

in the evident way; in the case of $\mathcal{O}$ *and* H *this is by means of* inc.

Definition C.8. A *closed strong* κ*-functor* from a closed strong κ-category $(\mathcal{C}, H, \mathcal{L}, \ldots)$ to another $(\mathcal{C}', H', \mathcal{L}', \ldots)$ consists of

- a function $\mathbf{Ob}\, F$ from $\mathbf{Ob}\, \mathcal{C}$ to $\mathbf{Ob}\, \mathcal{C}'$
- functions

$$\mathcal{C}(A,B) \xrightarrow{F^{\mathsf{v}}(A,B)} \mathcal{C}'(FA,FB) \qquad \text{for each } A,B \in \mathbf{Ob}\,\mathcal{C}$$

$$\mathcal{L}_A B \xrightarrow{F^{\mathsf{p}}(A,B)} \mathcal{L}'_{FA} FB \qquad \text{for each } A,B \in \mathbf{Ob}\,\mathcal{C}$$

$$H_A(B,C) \xrightarrow{F^{\mathsf{h}}(A,B)} H'_{FA}(FB,FC) \qquad \text{for each } A,B,C \in \mathbf{Ob}\,\mathcal{C}$$

- "coherence" morphisms in **Isos** $\mathcal{C}'$, up to which F preserves each object operation

$$\mathbf{1} \qquad F1 \xrightarrow{F^{1}} 1'$$

$$\times \qquad F(A \times B) \xrightarrow{F^{\times}_{AB}} FA \times' FB \qquad \text{for each } A,B \in \mathbf{Ob}\,\mathcal{C}$$

$$\to \qquad F(A \to B) \xrightarrow{F^{\to}_{AB}} FA \to' FB \qquad \text{for each } A,B \in \mathbf{Ob}\,\mathcal{C}$$

- such that F preserves, up to the coherence isomorphisms, each of the primitive operations listed in Lemma C.7. For example

  preservation of *

$$\begin{tikzcd}
\mathcal{C}(A,B) \times H_B(C,D) \arrow[r, "F^{\mathsf{v}}(A,B) \times F^{\mathsf{p}}_B(C,D)"] \arrow[d, "{}^*{}_{ABCD}"'] & \mathcal{C}(FA,FB) \times H'_{FB}(FC,FD) \arrow[d, "{}^{*\prime}{}_{(FA)(FB)(FC)(FD)}"] \\
H_A(C,D) \arrow[r, "F^{\mathsf{p}}_A(C,D)"'] & H'_{FA}(FC,FD)
\end{tikzcd}$$

  preservation of str

$$\begin{tikzcd}
\mathcal{L}_{A\times B} C \arrow[r, "F^{\mathsf{p}}_{A\times B} C"] \arrow[d, "\mathsf{str}_{ABC}"'] & \mathcal{L}'_{F(A\times B)} FC \arrow[r, "\mathcal{L}'_{F^{\times}_{AB}} FC"] & \mathcal{L}'_{FA\times FB} FC \arrow[d, "\mathsf{str}'_{(FA)(FB)(FC)}"] \\
H_A(B,C) \arrow[rr, "F^{\mathsf{h}}_A(B,C)"'] & & H'_{FA}(FB,FC)
\end{tikzcd}$$

Definition C.9. Let F, G be closed strong κ-functors from $(\mathcal{C}, H, \mathcal{L}, \ldots)$ to $(\mathcal{C}', H', \mathcal{L}', \ldots)$. A *closed strong κ natural isomorphism* from F to G is

- a family of morphisms in **Isos** $\mathcal{C}'$

$$FA \xrightarrow{\alpha A} GA$$

- preserving the homset operations i.e.

$$\begin{tikzcd}
\mathcal{C}(A,B) \arrow[d, "F^{\mathsf{v}}(A,B)"'] \arrow[dr, "G^{\mathsf{v}}(A,B)"] & \\
\mathcal{C}'(FA,FB) \arrow[r, "\mathcal{C}'(\alpha A, \alpha B)"'] & \mathcal{C}'(GA,GB)
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{L}_A B \arrow[d, "F^{\mathsf{p}}(A,B)"'] \arrow[dr, "G^{\mathsf{p}}(A,B)"] & \\
\mathcal{L}'_{FA} FB \arrow[r, "\mathcal{L}'_{\alpha A}\alpha B"'] & \mathcal{L}'_{GA} GB
\end{tikzcd}$$

$$\begin{tikzcd}
H_A(B,C) \arrow[d, "F^{\mathsf{h}}_A(B,C)"'] \arrow[dr, "G^{\mathsf{h}}_A(B,C)"] & \\
H'_{FA}(FB,FC) \arrow[r, "H'_{\alpha A}(\alpha B, \alpha C)"'] & H'_{GA}(GB,GC)
\end{tikzcd}$$

- and preserving the coherence isomorphisms i.e.

$$\begin{tikzcd}
F1 \arrow[r, "F^1"] \arrow[d, "\alpha 1"'] & 1' \arrow[d, "\mathsf{id}"] \\
G1 \arrow[r, "G^1"] & 1'
\end{tikzcd}
\qquad
\begin{tikzcd}
F(B \times C) \arrow[r, "F^{\times}_{BC}"] \arrow[d, "\alpha(B \times C)"'] & FB \times' FC \arrow[d, "\alpha B \times \alpha C"] \\
G(B \times C) \arrow[r, "G^{\times}_{BC}"'] & GB \times' GC
\end{tikzcd}$$

$$\begin{tikzcd}
F(B \to C) \arrow[r, "F^{\to}_{BC}"] \arrow[d, "\alpha(B \to C)"'] & FB \to' FC \arrow[d, "\alpha B \to' \alpha C"] \\
G(B \to C) \arrow[r, "G^{\to}_{BC}"'] & GB \to' GC
\end{tikzcd}$$