Electronic Notes in Theoretical Computer Science 40 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume40.html 14 pages

Models for the computational λ-calculus

John Power [1]

Laboratory for the Foundations of Computer Science, Division of Informatics
University of Edinburgh
King's Buildings, Edinburgh EH9 3JZ, SCOTLAND

Abstract

We consider several different sound and complete classes of models for the computational -calculus, explain the definitions, and outline why one might be interested in the various classes. We first consider the class of closed -categories, a natural and direct generalisation of the notion of cartesian closed category. We then consider closed -categories, which are based upon indexed categories and which are closely related to modern compiling technology. Finally, we consider the class of cartesian closed categories together with a -enriched monad. The latter class has the most developed abstract theory, which one can adopt and by which one can dispense with coherence details in the spirit of Mac Lane involving strengths.

1 Introduction

The computational λ-calculus, or λ_c-calculus, as introduced by Moggi in [13], 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 category C with finite products, and a strong monad T on C, such that T has Kleisli exponentials. One models a term in context by a map in $Kl(T)$, the Kleisli category for T. The class of λ_c-models is sound and complete, but it is not the only sound and complete class of models for the calculus, and there are advantages in considering some of the alternatives. Some of them are equivalent, with non-trivial proofs, to the class of λ_c-models, while others are not.

Here, we consider several sound and complete classes of models for the λ_c-calculus and outline some of the reasons why one may wish to consider them. Specifically, some sound and complete classes of models are given by

(i) λ_c-models
(ii) closed $Freyd$-categories

[1] This work is supported by EPSRC grant GR/M56333: The structure of programming languages: syntax and semantics

(iii) closed κ-categories, and

(iv) cartesian closed categories C together with a C-enriched monad on C.

In contrast to λ_c-models, closed $Freyd$-categories [20] provide a direct class of models for the λ_c-calculus. By that, we mean that, in a closed $Freyd$-category, a term of type X in context Γ is modelled by a map from the semantics of Γ to the semantics of X, whereas in a λ_c-model, it is modelled by a map from the semantics of Γ to the result of performing an operation, namely applying a monad, to the semantics of X. The perspective of closed $Freyd$-categories allows one, for instance, to give a natural notion of Henkin model [12] for the λ_c-calculus, which can be exploited in the study of data refinement [9].

Closed κ-categories [19,20,10] have a different character. They are given by indexed categories with extra data and axioms. The base category of the indexed category is where one models contexts and values, while one models computations in the fibres. This perspective relates more directly with modern compiling technology of functional programming languages [1] and suggests a decomposition of the λ_c-calculus into a version of the κ-calculus [6,20,10] together with *thunks*.

Cartesian closed categories C together with C-enriched monads eliminate the need for strengths. This is convenient in proving abstract mathematical results, as it is an instance of the substantial literature on enriched categories [8], allowing use of the much studied 2-category C-Cat of small C-enriched categories. That allows easy proofs of relationships between computational effects, for instance in seeing partiality as a part of continuations: one can consider a C-enriched map of monads from a monad for partiality to a monad for continuations, without concern at every step for the relationships between the strengths. This, when combined with axiomatic domain theory as developed for instance in [4] and [3], allows an account of recursion in the models. Perhaps more importantly, it is also the class of models best suited to modelling the operations associated with computational effects [15].

The notion of $Freyd$-category generalises that of category with finite products. The definition is delicate, composed of subsidiary definitions. First, a premonoidal category [18] is 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 $A' \otimes B'$ may differ. Such structures arise naturally in the presence of computational effects, where the difference between these two maps follows from sensitivity to evaluation order. It is routine to add a symmetry to a premonoidal category just as one does to a monoidal category [18]. We further say that a functor is the identity-on-objects if its object function is the identity function on a set. With these definitions, a $Freyd$-category consists of a symmetric premonoidal category K together with a category with finite products C and an identity-on-objects strict symmetric premonoidal functor

$J : C \longrightarrow K$. Closed $Freyd$-categories form a direct generalisation of the notion of cartesian closed categories.

In a κ-category, a term of type X in context Γ is modelled by an arrow of the form $1 \longrightarrow M(X)$ in a category that implicitly depends on Γ, more precisely, by an arrow from 1 to $M(X)$ in the fibre of the indexed category over $M(\Gamma)$. A κ-category has a weak first order notion of binding that amounts to the assertion that reindexing along projections has a left adjoint: there is coherence detail in the spirit of Mac Lane's book [11] missing in [19] and [20] but corrected in [10], most easily expressed by formulating the binding condition in terms equivalent to but not syntactically the same as that of adjointness subject to coherence. In programming terms, this corresponds to a form that binds an identifier but does not produce a *first class* function. This makes it suitable for compilation, as one may consider objects of the base category as stacks [1,20], with an additive approach to sequents.

Modelling the λ_c-calculus in terms of a cartesian closed category C together with a C-enriched monad T is essentially the same as modelling it in a λ_c-model: to give the enrichment of the monad is equivalent to giving the strength. However, this class of models is not equivalent to the others as we assume that C is cartesian closed, which is not the case in the other classes. However, it still provides a sound and complete class of models by Moggi's observation [14] that for completeness, it suffices to assume that the base category C is a topos, so in particular is cartesian closed. One benefit of this perspective is that C-Cat is a well studied 2-category [8], and in particular, the 2-category of monads in C-Cat has undergone extensive study [21]. One can exploit that abstract theory here by eliminating coherence details involving strengths [11].

The paper is organised as follows. In Section 2, we give a version of the λ_c-calculus: we do not use Moggi's original version in [13], but one with less redundancy, for convenience. In Section 3, we give the notion of $Freyd$-category and characterise the models of the λ_c-calculus as closed $Freyd$-categories. In Section 4, we define κ-categories, explain their relationship with the previous notions, and explain how they give rise to a decomposition of the λ_c-calculus that relates to current compilation technology. Finally, in Section 5, we give a sound and complete class of models in terms of enriched categories, and outline the abstract mathematical use of that perspective.

2 The computational λ-calculus

In this section, we give a version of the computational λ-calculus, or λ_c-calculus and recall Moggi's notion of λ_c-model [13]. There are several equivalent formulations of the λ_c-calculus. We shall not use the original formulation but one of the equivalent versions. This version of the λ_c-calculus has type constructors given by

$$X ::= B \mid X_1 \times X_2 \mid 1 \mid X \Rightarrow Y$$

where B is a base type and $X \Rightarrow Y$, rather than $X \to Y$, will be used to denote the exponential. We do not assert the existence of a type constructor TX: this formulation is equivalent to the original one because TX may be defined to be $1 \Rightarrow X$.

The terms of the λ_c-calculus are given by

$$e ::= x \mid b \mid e'e \mid \lambda x.e \mid * \mid (e, e') \mid \pi_i(e)$$

where x is a variable, b is a base term of arbitrary type, $*$ is of type 1, with π_i existing for $i = 1$ or 2, all subject to the evident typing. Again, this differs from the original formulation in that we do not explicitly have a *let* constructor or constructions $[e]$ or $\mu(e)$. Again, the two formulations are equivalent as we may consider $let\ x = e\ in\ e'$ as syntactic sugar for $(\lambda x.e')e$, and $[e]$ as syntactic sugar for $(\lambda x).e$ where x is of type 1, and $\mu(e)$ as syntactic sugar for $e(*)$.

We use this formulation as it has less data, allowing for easier proofs. Moreover, it is more directly a fragment of a typical call-by-value language: the above type constructors and, with the possible exception of π's, term constructors often appear explicitly in call-by-value languages, whereas T-types, μ, and $[-]$ typically do not.

The λ_c-calculus has two predicates, existence, denoted by $\downarrow$, and equivalence, denoted by $\equiv$. The $\downarrow$ rules may be expressed as saying $* \downarrow$, $x \downarrow$, $\lambda x.e \downarrow$ for all e, if $e \downarrow$ then $\pi_i(e) \downarrow$, and similarly for (e, e'). A value is a term e such that $e \downarrow$. There are two classes of rules for $\equiv$. The first class say that $\equiv$ is a congruence, with variables allowed to range over values. And the second class are rules for the basic constructions and for unit, product and functional types. It follows from the rules for both predicates that types together with equivalence classes of terms form a category, with a subcategory determined by values.

It is straightforward, using the original formulation of the λ_c-calculus in [13], to spell out the inference rules required to make this formulation agree with the original one: one just bears in mind that the models are the same, and we use syntactic sugar as detailed above.

The λ_c-calculus represents a fragment of a call by value programming language. In particular, it was designed to model fragments of ML, but is also a fragment of other languages such as the idealised language FPC introduced in [3]. For category theoretic models, the key feature is that there are two entities, expressions and values. So the most direct way to model the language as we have formulated it is in terms of a pair of categories V and E, together with an identity-on-objects inclusion functor $J : V \longrightarrow E$. This train of thought leads directly to the notion of closed $Freyd$-category, which we shall study in the next section. But the first sound and complete class of models for the λ_c-calculus was given by Moggi in [13].

For Moggi, a λ_c-model consists of a category C with finite products, together with a strong monad T on C, such that T has Kleisli exponentials, i.e., for each pair of objects X and Y, the functor $C(- \times X, TY) : C^{op} \longrightarrow Set$

is representable. In other words, there exists an object $X \Rightarrow Y$ such that $C(Z \times X, TY)$ is isomorphic to $C(Z, X \Rightarrow Y)$ for all Z, naturally in Z. A term of type X in context Γ is modelled by a map in the Kleisli category for T, i.e., by a map in C from $M(\Gamma)$ to $TM(X)$, where $M(-)$ denotes the semantic operation.

3 Closed *Freyd*-categories

In this section, we define and explain the notion of closed *Freyd*-category and give an indication of the perspective it gives on the λ_c-calculus.

We first recall the definitions of premonoidal category and strict premonoidal functor, and symmetries for them, as introduced in [18] and further studied in [17]. We use them to define the notion of *Freyd*-category. 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 $f : X \longrightarrow Y$ and $f' : X' \longrightarrow Y'$, the evident two maps from $X \otimes X'$ to $Y \otimes Y'$ may differ.

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

Definition 3.1 A *binoidal category* is a category K together with, for each object X of K, functors $h_X : K \longrightarrow K$ and $k_X : K \longrightarrow K$ such that for each pair (X, Y) of objects of K, $h_X Y = k_Y X$. The joint value is denoted $X \otimes Y$.

Definition 3.2 An arrow $f : X \longrightarrow X'$ in a binoidal category K is *central* if for every arrow $g : Y \longrightarrow Y'$, the following diagrams commute

$$\begin{array}{ccc} X \otimes Y & \xrightarrow{X \otimes g} & X \otimes Y' \\ {\scriptstyle f \otimes Y}\downarrow & & \downarrow{\scriptstyle f \otimes Y'} \\ X' \otimes Y & \xrightarrow[X' \otimes g]{} & X' \otimes Y' \end{array} \qquad \begin{array}{ccc} Y \otimes X & \xrightarrow{g \otimes X} & Y' \otimes X \\ {\scriptstyle Y \otimes f}\downarrow & & \downarrow{\scriptstyle Y' \otimes f} \\ X \otimes X' & \xrightarrow[g \otimes X']{} & Y' \otimes X' \end{array}$$

A natural transformation $\alpha : G \Longrightarrow H : C \longrightarrow K$ is called *central* if every component of α is central.

Definition 3.3 A *premonoidal category* is a binoidal category K together with an object I of K, and central natural isomorphisms a with components $(X \otimes Y) \otimes Z \longrightarrow X \otimes (Y \otimes Z)$, l with components $X \longrightarrow X \otimes I$, and r with components $X \longrightarrow I \otimes X$, subject to two equations: the pentagon expressing coherence of a, and the triangle expressing coherence of l and r with respect to a (see [8] for an explicit depiction of the diagrams).

Proposition 3.4 *Given a strong monad T on a symmetric monoidal category C, the Kleisli category $Kl(T)$ for T is a premonoidal category, with the functor $J : C \longrightarrow Kl(T)$ preserving premonoidal structure strictly: a monoidal category such as C is trivially a premonoidal category.*

So every λ_c-model gives rise to a premonoidal category.

Definition 3.5 Given a premonoidal category K, the *centre* of K, denoted $Z(K)$, is the subcategory of K consisting of all the objects of K and the central morphisms.

Given a strong monad on a symmetric monoidal category, the base category C need not be the centre of $Kl(T)$. But, modulo the condition that $J : C \longrightarrow Kl(T)$ be faithful, or equivalently, the mono requirement [13,18], i.e., the condition that the unit of the adjunction be pointwise monomorphic, it must be a subcategory of the centre.

The functors h_X and k_X preserve central maps. So we have

Proposition 3.6 *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 [18].

Definition 3.7 A *symmetry* for a premonoidal category is a central natural isomorphism with components $c : X \otimes Y \longrightarrow Y \otimes X$, satisfying the two conditions $c^2 = 1$ and equality of the evident two maps from $(X \otimes Y) \otimes Z$ to $Z \otimes (X \otimes Y)$. A *symmetric* premonoidal category is a premonoidal category together with a symmetry.

Symmetric premonoidal categories are those of primary interest to us, and seem to be those of primary interest in denotational semantics in general.

Definition 3.8 A *strict premonoidal functor* is a functor that preserves all the structure and sends central maps to central maps.

One may similarly generalise the definition of strict symmetric monoidal functor to strict symmetric premonoidal functor.

Definition 3.9 A *Freyd*-category consists of a category C with finite products, a symmetric premonoidal category K, and an identity-on-objects strict symmetric premonoidal functor $J : C \longrightarrow K$. A strict *Freyd*-functor consists of a pair of functors that preserve all the *Freyd*-structure strictly.

Definition 3.10 A *Freyd*-category $J : C \longrightarrow K$ is *closed* if for every object X, the functor $J(X \otimes -) : C \longrightarrow K$ has a right adjoint. A strict *closed Freyd*-functor is a *Freyd*-functor that preserves all the closed structure strictly.

Observe that it follows that the functor $J : C \longrightarrow K$ has a right adjoint, and so K is the Kleisli category for a monad on C. We sometimes write K for a $Freyd$-category, as the rest of the structure is usually implicit: often, it is given by $Z(K)$ and the inclusion.

A variant of one of the main theorems of [17] is

Theorem 3.11 *To give a closed $Freyd$-category is to give a category C with finite products together with a strong monad T on C together with assigned Kleisli exponentials. To give a strict closed $Freyd$-functor is to give a strict map of strong monads that strictly preserves Kleisli exponentials.*

Given a category C with finite products and a strong monad T on it, $Kl(T)$ is a $Freyd$-category. However, although a functor preserving the strong monad and the finite products yields a strict $Freyd$-functor, the converse is not true.

It follows from Moggi's result, but may also be proved directly, that closed $Freyd$-categories provide a sound and complete class of models for the λ_c-calculus. It is routine to spell out how the λ_c-calculus as we described it in Section 2 is modelled directly in a closed $Freyd$-category: the structure in the definition of closed $Freyd$-category mirrors almost exactly the structure of the definition of the calculus. The one point to note is that one must make a decision how to model (e, e') as in principle there are two possibilities of which one must make an arbitrary choice. So we choose to model $\Gamma \vdash (e, e') : X \times Y$ by the composite

$$M(\Gamma) \longrightarrow M(\Gamma) \times M(\Gamma) \longrightarrow M(X) \times M(\Gamma) \longrightarrow M(X) \times M(Y)$$

with the evident maps given by the $Freyd$-structure. The λ-terms and application are modelled directly using the closed structure.

There are several uses of this formulation of models of the λ_c-calculus. It provides a direct class of models, in the sense that a term in context $\Gamma \vdash t : X$ is modelled by a map in the $Freyd$-category from $M(\Gamma)$ to $M(X)$. This fits well with the classes of models of computational phenomena given by games models, as explained in [2] and [7].

This formulation may also be used to generalise the notion of Henkin model [12] from the simply typed λ-calculus to the λ_c-calculus. For the simply typed λ-calculus, one has

Proposition 3.12 *If L is the cartesian closed category freely generated by a signature for the λ-calculus, then to give a Henkin model is equivalent to giving a finite product preserving functor $H : L \longrightarrow Set$ such that the induced functions $H(\sigma \Rightarrow \tau) \longrightarrow (H(\sigma) \Rightarrow H(\tau))$ are injective.*

A variant of this proposition has been used in analysing data refinement for the simply typed λ-calculus: one needs to relax the notion of logical relation to a notion of lax logical relation, in order to allow composition of data refinements [16], and that relaxation is exactly in the sense of Proposition 3.12. Using the notion of $Freyd$-category, we can generalise this analysis from the

simply typed λ-calculus to the λ_c-calculus as follows.

Definition 3.13 If L is the closed $Freyd$-category freely generated by a signature for the λ_c-calculus, a *Henkin* model is a $Freyd$-functor from L to the $Freyd$-category given by the inclusion $J : Set \longrightarrow Set_p$ such that the induced functions $H(\sigma \Rightarrow \tau) \longrightarrow (H(\sigma) \Rightarrow H(\tau))$ are injective, where Set_p is the category of small sets and partial functions.

A binary version of this, meaning the systematic replacement of a set by a pair of sets, has been used to give an account of data refinement for the λ_c-calculus [9]. We observed above that $Freyd$-functors cannot be expressed in terms of the structure of a λ_c-model. So this provides one instance of the value of considering closed $Freyd$-categories as models of the λ_c-calculus.

4 Closed κ-categories

In this section, we introduce κ-categories, as defined in [19] and modified and slightly corrected in [20] and especially [10], with the idea of using closed κ-categories as models for the λ_c-calculus. It was shown in [19] that $Freyd$-categories are equivalent to κ-categories: we recall the construction here. It is routine to extend that equivalence to see that closed $Freyd$-categories are equivalent to closed κ-categories, showing that the latter provide a sound and complete class of models for the λ_c-calculus [20].

Given a small category C, a functor from C^{op} to Cat is called an *indexed category*, a natural transformation between two indexed categories is called an *indexed functor*. The notion of *indexed natural transformation* is definable too.

Definition 4.1 A *κ-category* consists of a small category C with finite products, together with an indexed category $H : C^{op} \longrightarrow Cat$ such that

- for each object A of C, $ObH_A = ObC$, and for each arrow $f : A \longrightarrow B$ in C, the functor $H_f : H_B \longrightarrow H_A$ is an identity-on-objects functor
- for every triple of objects, there is an isomorphism $\kappa : H_{A\times B}(1, H_\pi D) \longrightarrow H_A(B, D)$ natural in A and D.
- the two functions from $H_1(1, D)$ to $H_{1\times 1}(1, D)$, one given by reindexing and the other given by κ, are equal.

Observe from the definition that for each projection $\pi : B \times A \longrightarrow B$ in 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

$$\kappa : H_{B\times A}(D, D') \cong H_B(D \times A, D').$$

Definition 4.2 A κ-category $H : C^{op} \longrightarrow Cat$ is *closed* if for every object A of C, every object has a generic map from A into it, i.e., for every object B

of C, there is an object $[A, B]$ and a map $\mathsf{apply} : A \longrightarrow B$ in $H_{[A,B]}$ such that for every object X and every map $f : A \longrightarrow B$ in H_X, there exists a unique map $\lambda f : X \longrightarrow [A, B]$ in C such that $H_{\lambda f}(\mathsf{apply}) = f$.

To see how $Freyd$-categories give rise to κ-categories, we construct one of the latter from one of the former. This requires some supplementary definitions.

Definition 4.3 A *comonoid* in a premonoidal category K consists of an object A of K, and central maps $\delta : A \longrightarrow A \otimes A$ and $\nu : A \longrightarrow I$ making the usual associativity and unit diagrams commute. A *comonoid map* from A to B in a premonoidal category K is a central map $f : A \longrightarrow B$ that commutes with the comultiplications and counits of the comonoids.

Given a premonoidal category K, comonoids and comonoid maps in K form a category $Comon(K)$ with composition given by that of K. Moreover, any strict premonoidal functor $H : K \longrightarrow L$ lifts to a functor $Comon(H) : Comon(K) \longrightarrow Comon(L)$.

Trivially, any comonoid A yields a comonad $- \otimes A$, and any comonoid map $f : A \longrightarrow B$ yields a functor from $Kl(- \otimes A)$, the Kleisli category of the comonad $- \otimes A$, to $Kl(- \otimes B)$, that is the identity on objects. So we have a functor $simple(K) : Comon(K)^{op} \longrightarrow Cat$.

Given a category C with finite products, every object A of C has a unique comonoid structure, given by the diagonal and the unique map to the terminal object. So $Comon(C)$ is isomorphic to C.

Thus, given a $Freyd$-category $J : C \longrightarrow K$, we have a functor given by composing $simple(K)$ with the functor induced by J from $C \cong Comon(C)$ to $Comon(K)$: we denote this composite functor by $s(J) : C^{op} \longrightarrow Cat$.

Theorem 4.4 *[19] For any $Freyd$-category $J : C \longrightarrow K$, the indexed functor $s(J) : C^{op} \longrightarrow Cat$ is a κ-category, and every κ-category arises uniquely up to coherent isomorphism from a $Freyd$-category.*

Corollary 4.5 *For any closed $Freyd$-category J, the κ-category $s(J)$ is closed, and for any closed κ-category H, there is a closed $Freyd$-category J unique up to coherent isomorphism such that $s(J)$ is isomorphic to H.*

This result means that closed κ-categories provide a sound and complete class of models for the λ_c-calculus. The finite products in the base category show how to model product types; the definition of the closed structure shows how to model higher order types. A term of type X in context Γ is modelled by a map from 1 to $M(X)$ in the fibre over $M(\Gamma)$. The structure of the definition of closed κ-category shows how to model the term-constructors. To model (e, e'), follow the instructions of Section 3 with the evident modification from $Freyd$-structure to κ-structure.

Modelling the simply typed λ-calculus with indexed categories highlights

different features of λ-calculus to modelling it with cartesian closed categories. Similarly here, modelling the λ_c-calculus with closed κ-categories highlights different features of it to those highlighted by closed *Freyd*-categories. For κ-categories, one may see the base category as providing stacks. One can see the definition of a κ-category as giving rise to a syntax that is remarkably similar to a class of closely related sets of syntax recently being used in compiler technology for functional programming languages [1]. So κ-categories and the results surrounding them provide semantic support for current work on compilers, as well as suggesting a decomposition of the λ_c-calculus that we might call the κ-calculus [20] owing to its relationship with Hasegawa's notion [6].

As λ_c-models are equivalent to closed κ-categories, so correspondingly, the terms of the λ_c-calculus are given by adding *thunks* to the κ-calculus. Predicates and rules for the κ-calculus can be derived from those for a first order fragment of the λ_c-calculus.

A computation judgement of the κ-calculus is of the form

$$x_1 : C_1, \ldots, x_n : C_n, A \vdash_c M : B$$

where A is a type, understood to be the type of the stack before M is run, with B being the type of the stack after M is run.

A value judgement is of the form

$$x_1 : C_1, \ldots, x_n : C_n \vdash V : B$$

A computation judgement denotes a morphism from A to B in the fibre over $C_1 \times \cdots \times C_n$. A value judgement denotes a morphism in the base in the usual fashion. The syntax and the key typing rules are as follows:

$$\frac{\Gamma \vdash V : C}{\Gamma, A \vdash_c push(V) : C \times A} \qquad \frac{\Gamma, x : C, A \vdash_c M : B}{\Gamma, C \times A \vdash_c \kappa x.M : B}$$

$$\frac{\Gamma, A \vdash_c M : B \quad \Gamma, B \vdash_c N : C}{\Gamma, A \vdash_c M; N : C}$$

Adding *thunks* allows one to recover a calculus equivalent to the λ_c-calculus. The typing required is

$$\frac{\Gamma, A \vdash_c M : B}{\Gamma \vdash \texttt{mkthunk}\, M : [A \to B]} \qquad \frac{}{A \times [A \to B] \vdash_c \texttt{apply} : B}$$

So $push(V)$ can be read as pushing a value on the stack (the top of the stack is written on the left), while $\kappa x.M$ pops a value and binds it to x in M. The symbol ; is modelled by composition in the fibre of the indexed category over $M(\Gamma)$. The idea here is that the most recently pushed value is popped by κ and, upon adding the assumption of closedness, `apply` opens a closure made by `mkthunk`. For more analysis of this calculus, see [20] and [10].

5 Enriched categories

In this section, we outline our last class of models for the λ_c-calculus.

For any cartesian closed (or more generally symmetric monoidal closed) category C, one can speak of a C-category. The idea is that instead of having homsets as in the usual definition of category, a C-category has a homobject of C. Formally, the definition is as follows.

Definition 5.1 A C-category D consists of

- a set ObD, elements of which are called objects of D,
- for each pair (A, B) of objects of D, an object $D(A, B)$ of C
- for each triple (A, B, E), a map $\circ : D(B, E) \times D(A, B) \longrightarrow D(A, E)$
- for each object A, a map $j_A : 1 \longrightarrow D(A, A)$

subject to equations to force *comp* to be associative with identities given by the j_A's.

The leading example has $C = Set$, in which case a C-category is exactly a locally small category in the usual way. Other examples of primary interest in denotational semantics have $C = Poset$ or the category of ω-cpo's, or a presheaf category, or more generally a topos. One can also consider $C = Cat$.

Definition 5.2 A C-functor $H : D \longrightarrow E$ consists of

- a function $ObH : ObD \longrightarrow ObE$, which we typically abbreviate to H
- for each pair (A, B) of objects of D, a map $H_{(A,B)} : D(A, B) \longrightarrow E(HA, HB)$

subject to axioms to the effect that H preserves composition and identities.

The notion of C-natural transformation can be made in a similar vein (see [8] for details), giving rise to notions of C-monad and C-Kleisli category. This is most elegantly seen as an instance of Street's analysis of monads in a 2-category, where one considers the 2-category C-Cat [21].

It is folklore, but also appears as a (very) special case of the main result of [5] that we have

Theorem 5.3 *To give a strong monad on a cartesian closed category C is equivalent to giving a C-monad on C.*

Proof. Given a strong monad T on C, we need to define maps in C of the form $(A \Rightarrow B) \longrightarrow (TA \Rightarrow TB)$. In order to do that, by the Yoneda lemma, it suffices to give a natural family of functions

$$C(X, A \Rightarrow B) \longrightarrow C(X, TA \Rightarrow TB)$$

but $C(X, A \Rightarrow B)$ is isomorphic to $C(X \times A, B)$ and similarly for $C(X, TA \Rightarrow TB)$. So the data for a strength gives such functions, and the axioms yield the axioms for an enriched functor. The converse construction also uses the Yoneda lemma, but uses naturality in B rather than in X. It is a tedious calculation involving nested applications of the Yoneda lemma to verify that the axioms for a strength match those for an enrichment. □

It follows from Moggi's observation that strong monads on toposes give a sound and complete class of λ_c-models that the class of cartesian closed categories C together with a C-monad gives another sound and complete class of λ_c-models, albeit one that is not equivalent to those we have considered so far.

The modelling of the λ_c-calculus in such a structure does not seem to tell us much that is new about the λ_c-calculus directly: one just copies the usual modelling in a λ_c-model. There is a established notion of a *tensored* C-category [8] that may provide insight: all C-categories of the form $Kl(T)$ for a C-monad T on C are tensored, and the tensor models products of types.

However, this perspective does allow abstract mathematical results of relevant interest. For instance, it follows immediately from this characterisation that any monad on Set has precisely one strength: to give a strength is equivalent to giving an enrichment, and a monad on Set is trivially and uniquely enriched over Set.

Similarly, it also follows immediately that any monad on $Poset$ or the category of ω-cpo's has at most one strength, so again, one need not bother looking for more once one has found one. In fact, there is at most one enrichment of a monad on Cat too, but that fact is not trivial.

Street's analysis [21] of monads in 2-categories, such as the 2-category C-Cat of small C-categories, yields an abstract analysis of the relationships between computational effects. For instance, it shows

Theorem 5.4 *Given C-monads S and T on a cartesian closed category C, to give a C-functor $H : Kl(S) \longrightarrow Kl(T)$ commuting with the canonical C-functors from C is equivalent to giving a C-monad morphism from S to T.*

Unwinding that result, it follows from the definition of tensor [8] that any C-functor as in the theorem preserves tensors, and consequently, to give such a C-functor is equivalent to giving a strict $Freyd$-functor. The notion of C-monad morphism is definitive, as illustrated by Street's theory, so using the equivalence between C-monads and monads with a strength, it gives a definitive notion of a map between a pair of monads with strengths: it amounts to a monad morphism that respects the strengths. Putting that together, we conclude

Corollary 5.5 *Given strong monads S and T on a cartesian closed category C, to give a strict $Freyd$-functor $H : Kl(S) \longrightarrow Kl(T)$ commuting with the canonical functors from C is equivalent to giving a morphism of strong monads from S to T.*

One reason that is of interest is if one takes S to be a monad for partiality, then one needs a strict $Freyd$-functor from $Kl(S)$ to $Kl(T)$ in order to incorporate an account of recursion into the λ_c-calculus with models taken in terms of the strong monad T.

But the sound and complete class of models of this section has not been fully utilised yet. It certainly provides a foundation for modelling the algebraic operations associated with computational effects [15], but it may in fact suggest a more primitive class of models in terms of operations and equations or perhaps, even more primitively, in terms of operations and observations.

6 Conclusions

Eugenio Moggi introduced the computational λ-calculus, or λ_c-calculus, in [13], in order to support a unified treatment of computational effects. He gave a sound and complete class of models for the λ_c-calculus based upon a category C with finite products and a strong monad T on C such that T has Kleisli exponentials; then he modelled the λ_c-calculus in the Kleisli category $Kl(T)$ of T.

Here, we have presented three other sound and complete classes of models for the λ_c-calculus, some of them equivalent to the class of models presented by Moggi, and one of them inequivalent. For different purposes, it seems best to use different classes; each of the classes generalises naturally in a different way to the rest. For instance, closed $Freyd$-categories [18,19] provide direct models. Closed κ-categories [19,10], in contrast, generalise naturally in the direction of dependent type theories. And the use of enriched categories seems most natural when one wants to extend the modelling of the λ_c-calculus to modelling algebraic operations associated with computational effects [15], such as nondeterministic *or*.

References

[1] R. Douence and P. Fradet, *A Taxonomy of Functional Language Implementations Part I : Call-by-Value*, INRIA Research Report No 2783, 1996.

[2] M. Fiore and K. Honda, *Recursive Types in Games: Axiomatics and Process Representation*, Proc LICS **98** (1998) 345–356.

[3] M. Fiore and G.D. Plotkin, em An axiomatisation of computationally adequate domain-theoretic models of , Proc LICS **94** (1994) 92–102.

[4] M. Fiore, G.D. Plotkin, and A.J. Power, *Complete cuboidal sets in Axiomatic Domain Theory*, Proc LICS **97** (1997) 268–279.

[5] R. Gordon and A.J. Power, *Enrichment through variation*, J. Pure Appl. Algebra **120** (1997) 167–185.

[6] M. Hasegawa, *Decomposing typed lambda calculus into a couple of categorical programming languages*, "Proc. CTCS 95," Lect. Notes in Computer Science **953**, Springer (1995).

[7] K. Honda and N. Yoshida, *Game-theoretic analysis of call-by-value computation*, "Proc ICALP 97," Lect. Notes in Computer Science **1256** (1997) 225–236.

[8] G.M. Kelly, "Basic concepts of enriched category theory," Cambridge University Press (1982).

[9] Y. Kinoshita and A.J. Power, *Data refinement for a call-by-value language*, "Proc CSL 99," Lect. Notes in Computer Science **1683** (1999) 562–576.

[10] P.B. Levy, A.J. Power, and H. Thielecke, *Modelling environments in call-by-value programming languages* (submitted).

[11] Saunders Mac Lane, "Categories for the Working Mathematician," Springer-Verlag (1971).

[12] J. Mitchell, "Foundations for programming languages," Foundations of Computing Series, MIT Press (1996).

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

[14] E. Moggi, *Notions of computation and Monads*, Information and Computation **93** (1991) 55–92.

[15] G.D. Plotkin and A.J. Power, *Semantics for Algebraic Operations*, "Proc MFPS 01," Electronic Notes in Theoret. Computer Science **45** (2001).

[16] G.D. Plotkin, A.J. Power, D.T. Sannella, and R.D. Tennent, *Lax logical relations*, "Proc ICALP 2000," Lect. Notes in Computer Science **1853** (2000) 85–102.

[17] A.J. Power, *Premonoidal categories as categories with algebraic structure*, Theoretical Computer Science (to appear).

[18] A.J. Power and E.P. Robinson, *Premonoidal categories and notions of computation*, Math Structure in Computer Science **7** (1997) 453–468.

[19] A.J. Power and H. Thielecke, *Environments, Continuation Semantics and Indexed Categories*, "Proc TACS 97," Lect. Notes in Computer Science **1281** (1997) 391–41.

[20] A.J. Power and H. Thielecke, *Closed - and -categories*, "Proc ICALP 99," Lect. Notes in Computer Science **1644** (1999) 625–634.

[21] Ross Street, *The formal theory of monads*, J. Pure Appl. Algebra **2** (1972) 149–168.