Closed Freyd- and κ-categories

John Power[1*] and Hayo Thielecke[2**]

1 University of Edinburgh, Edinburgh EH9 3JZ, Scotland.
2 QMW, University of London, London E1 4NS, UK.

Abstract. We give two classes of sound and complete models for the computational λ-calculus, or λ_c-calculus. For the first, we generalise the notion of cartesian closed category to that of closed Freyd-category. For the second, we generalise simple indexed categories. The former gives a direct semantics for the computational λ-calculus. The latter corresponds to an idealisation of stack-based intermediate languages used in some approaches to compiling.

1 Introduction

The computational λ-calculus, or λ_c-calculus, 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 small category $\mathcal{C}$ with finite products, and a strong monad $\mathcal{T}$ on $\mathcal{C}$, such that $\mathcal{T}$ has Kleisli exponentials. 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 X in context Γ is not modelled by an arrow in $\mathcal{C}$ from the semantics of Γ to the semantics of X, but by a derived construction in terms of the monad.

In this paper, we give two different equivalent formulations of λ_c-models, one generalising the notion of cartesian closed category, the other given by an indexed category with structure. We investigate natural languages for these structures, and we prove the various equivalences. This builds upon our work in [11], in which we proved an equivalence between premonoidal categories with extra structure (now called *Freyd*-categories) and indexed categories with added structure, which we called κ-categories. We include an appendix recalling the definitions and relationship.

Although these two new formulations are equivalent to λ_c-models, they are not the same as them. That is, the primitives in each are different. We explore this point by formulating languages motivated by each of the two models. One of the languages emphasises the role of structural rules of weakening and contraction, while the other emphasises the sequencing of intermediate values (inherent in the `let` of λ_c) by keeping an explicit stack.

A Freyd-category consists of a symmetric premonoidal category $\mathcal{K}$ together with a category with finite products $\mathcal{C}$ and an identity on objects strict symmetric

[*] This work is supported by EPSRC grant GR/J84205: Frameworks for programming language semantics and logic.
[**] Supported by EPSRC grant GR/L54639.

premonoidal functor $J:\mathcal{C} \longrightarrow \mathcal{K}$, where 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 $A' \otimes B'$ 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. A program phrase in environment Γ is modelled by a morphism in the premonoidal category with domain $[\![\Gamma]\!]$. The semantics here suggests a multiplicative approach to sequents.

In a κ-category, a program phrase in environment Γ is modelled by an element $\mathbf{1} \longrightarrow [\![B]\!]$ in a category that implicitly depends on Γ, i.e., by an arrow from $\mathbf{1}$ to $[\![B]\!]$ in the fibre of the indexed category over $[\![\Gamma]\!]$. A κ-category has a weak first-order notion of binding, given by the assertion that reindexing along projections has a left adjoint. In programming terms, that corresponds to a special form that binds an identifier but is not reifying in the sense that it does not produce a *first-class* function. In Hasegawa's original account, the κ-calculus was motivated purely by categorical considerations. However, such a first-order binder was independently devised by Douence and Fradet to give a high-level account of stack operations in compiling [1].

In [11], we gave an equivalence between Freyd-categories and κ-categories. Here, we extend that equivalence to model 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 computations. This leads us to a notion of closedness for a Freyd-category [9]. To give a closed Freyd-category is equivalent to giving a strong monad with Kleisli exponentials on a cartesian category. But to give a strong monad on a cartesian category is to give an ordinary monad on the associated κ-category, where the cartesian category is regarded as a degenerate symmetric premonoidal category. So we examine the relationships between all four notions: a closed Freyd-category, a strong monad with Kleisli exponentials on a cartesian category, a monad on a simple fibration, and a closed κ-category.

The paper is organised as follows. We extend our equivalence between Freyd-categories and κ-categories to one incorporating higher-order structure in Section 2. In Section 3, we consider two calculi designed to fit the primitives of these two categorical settings: one a slight variant of λ_c, the other an extension of Hasegawa's κ-calculus. Section 4 concludes.

2 Closed Freyd-categories, closed κ-categories, and λ_c-models

In previous work [11], recalled in Appendix A, we considered two ways of modelling environments. In this section, we extend that to model higher-order structure, allowing us two ways to model the λ_c-calculus [7]. We define and relate closed Freyd-categories and closed κ-categories with λ_c-models.

Definition 1. *A strong monad T on a cartesian category $\mathcal{C}$ has* Kleisli exponentials *if for every object A, the functor $J(A \times -): \mathcal{C} \longrightarrow$ **Kleisli**(T) has a right adjoint.*

Observe that if a cartesian category $\mathcal{C}$ is closed, then every strong monad on it has Kleisli exponentials. The converse is almost but not quite true.

Definition 2. *A Freyd-category $J: \mathcal{C} \longrightarrow \mathcal{K}$ is* closed *if for every object A, the functor $J(A \otimes -): \mathcal{C} \longrightarrow \mathcal{K}$ has a right adjoint.*

Observe that it follows that the functor $J: \mathcal{C} \longrightarrow \mathcal{K}$ has a right adjoint, and so $\mathcal{K}$ is the Kleisli category for a monad on $\mathcal{C}$. A variant of one of the main theorems of [9] is

Theorem 1. *To give a closed Freyd-category is to give a cartesian category together with a strong monad on it for which Kleisli exponentials exist.*

That is as good a result as one can imagine to relate closed Freyd-categories with strong monads. So now we turn to κ-categories.

Definition 3. *A κ-category $H: \mathcal{C}^{\mathrm{op}} \longrightarrow \mathcal{C}at$ is* closed *if for every object A of $\mathcal{C}$, every object has a generic map from A into it, i.e., for every object B of $\mathcal{C}$, there is an object $[A \to B]$ and a map $\mathsf{apply}: A \longrightarrow B$ in $H_{[A \to B]}$ such that for every object C and every map $f: A \longrightarrow B$ in H_C, there exists a unique map $\Lambda f: C \longrightarrow [A \to B]$ in $\mathcal{C}$ such that $H_{\Lambda f}(\mathsf{apply}) = f$.*

$$\begin{tikzcd} C \arrow[r, "\Lambda f"] & {[A \to B]} \end{tikzcd}$$

$$\begin{tikzcd} A \arrow[d, "f"', ""{name=F}] & & A \arrow[d, "\mathsf{apply}", ""'{name=G}] \\ B & & B \arrow[from=G, to=F, mapsto, "H_{\Lambda f}"'] \end{tikzcd}$$

Using the notation of Appendix A, we have

Proposition 1. *For any closed Freyd-category J, the κ-category $\mathsf{s}(J)$ is closed, and for any closed κ-category H, there is a Freyd-category J unique up to coherent isomorphism such that $\mathsf{s}(J)$ is isomorphic to H.*

Proof. Most of this follows from Theorem 4 in the appendix. Close inspection of the construction of $\mathsf{s}(J)$ yields the rest: the assertion of the existence of a generic map from any object C into any object A is routinely equivalent to the assertion that the corresponding Freyd-category is closed.

Relating this to monads, it was remarked by Plotkin and shown in [8] that

Proposition 2. *To give a strong monad with Kleisli exponentials on a cartesian category $\mathcal{C}$ is to give an indexed monad on the indexed category $\mathsf{s}(\mathcal{C})$.*

Finally, just as one can build the Kleisli construction from a monad on an ordinary category, one can build the Kleisli construction from a monad on an indexed category, in particular on a κ-category; and there is an equivalence between monads and identity on objects indexed functors that have indexed right adjoints. Thus we have

Proposition 3. *To give a monad on* $\mathsf{s}(\mathcal{C})$ *for a cartesian category* $\mathcal{C}$ *is to give a* κ*-category* H *such that* $\mathsf{inc}\colon \mathsf{s}(\mathcal{C}) \longrightarrow H$ *has an indexed right adjoint.*

Putting this together, we have

Theorem 2. *Given a cartesian category* $\mathcal{C}$*, the following are equivalent:*

1. *to give a strong monad with Kleisli exponentials on* $\mathcal{C}$
2. *to give a closed Freyd-category* $J\colon \mathcal{C} \longrightarrow \mathcal{K}$
3. *to give a monad on* $\mathsf{s}(\mathcal{C})$ *subject to a closedness condition*
4. *to give a* κ*-category* $H\colon \mathcal{C}^{op} \longrightarrow Cat$ *such that* $\mathsf{inc}\colon \mathsf{s}(\mathcal{C}) \longrightarrow H$ *has an indexed right adjoint.*

The key result about the λ_c-calculus in [7] is that it has a sound and complete class of models given by the λ_c-models. But in Theorem 2, we have characterised λ_c-models. So one has

Theorem 3. *Both closed Freyd-categories and closed* κ*-categories form sound and complete classes of models for the computational* λ*-calculus.*

3 λ- and κ-calculi

Because of the categorical equivalence results in Section 2, we know that we can model the computational λ-calculus in closed Freyd-categories and closed κ-categories. Our aim here is not to study completeness of calculi; that is routine, given our analysis in the last section and the previous analysis by Moggi. Rather, we want to formulate languages that correspond more directly to our models in an effort to expose their different character. Furthermore, at least one of these languages appears to have independent interest [1]. Following our categorical equivalence result, we connect them by a stack-passing transformation.

3.1 Computational λ-calculus and Freyd-categories

In this section, we sketch how closed Freyd-categories give semantics for the λ_c-calculus. We already know, by Theorem 2, that the λ_c-calculus has a sound and complete class of models given by closed Freyd-categories, but the point here is to see how the primitives of the calculus as we will express it correspond to the primitives of the definition of closed Freyd-category.

Raw terms are given by the following grammar:

$$\begin{aligned} V &::= x \mid () \mid (V,V) \mid \lambda x.M \\ p &::= x \mid () \mid (x,x) \\ M &::= V \mid \mathtt{let}\, p = M\, \mathtt{in}\, M \mid VV \end{aligned}$$

Though this is not strictly necessary, we chose here a calculus in which a pair (M, N) would have to be explicitly sequenced, as it were, by writing

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

(or, if one wants right-to-left evaluation order, $\texttt{let}\, y = N \,\texttt{in}\,\texttt{let}\, x = M \,\texttt{in}\,(x, y)$). This is meant to emphasize that, in a *pre*monoidal setting, we need to take a little more care than in a monoidal one.

We first consider a first order fragment of the λ_c-calculus. Like monoidal categories, premonoidal categories naturally correspond to calculi with multiplicative contexts.

$$\frac{\Gamma \vdash N\colon A \quad \Delta, x\colon A \vdash M\colon B}{\Gamma, \Delta \vdash \texttt{let}\, x = N \,\texttt{in}\, M\colon B} \qquad \frac{\Gamma \vdash N\colon \texttt{unit} \quad \Delta \vdash M\colon B}{\Gamma, \Delta \vdash \texttt{let}\, () = N \,\texttt{in}\, M\colon B}$$

$$\frac{\Gamma \vdash N\colon A_1 {*} A_2 \quad \Delta, x_1\colon A_1, x_2\colon A_2 \vdash M\colon B}{\Gamma, \Delta \vdash \texttt{let}\,(x_1, x_2) = N \,\texttt{in}\, M\colon B}$$

The `let`-construct of the computational λ-calculus is interpreted by sequential composition (together with the premonoidal tensor for concatenation of contexts).

In a *symmetric* premonoidal category [10], we can permute variables in the sequents:

$$\frac{\Gamma, x\colon A, y\colon B, \Delta \vdash M\colon C}{\Gamma, y\colon B, x\colon A, \Delta \vdash M\colon C}$$

Semantically, this rule is interpreted by precomposition with $[\![\Gamma]\!] \otimes c \otimes [\![\Delta]\!]$, where c is the symmetry isomorphism $[\![B]\!] \otimes [\![A]\!] \cong [\![A]\!] \otimes [\![B]\!]$.

Furthermore, for a Freyd-category $J\colon \mathcal{C} \longrightarrow \mathcal{K}$ (Definition 4), we can add weakening and contraction in sequents

$$\frac{\Gamma \vdash M\colon B}{\Gamma, x\colon A \vdash M\colon B} \qquad \frac{\Gamma, x\colon A, y\colon A \vdash M\colon C}{\Gamma, x\colon A \vdash M[y := x]\colon C}$$

Weakening and contraction are interpreted by the counit $[\![\Gamma]\!] \otimes [\![A]\!] \longrightarrow [\![\Gamma]\!]$ and comultiplication $[\![\Gamma]\!] \otimes [\![A]\!] \longrightarrow [\![\Gamma]\!] \otimes [\![A]\!] \otimes [\![A]\!]$ of the comonad as described in Definition 5.

And as we have finite products in $\mathcal{C}$, we have additive sequents for first-order values:

$$\frac{}{x_1\colon B_1, \ldots, x_n\colon B_n \vdash x_i\colon B_i} \qquad \frac{}{\Gamma \vdash ()\colon \texttt{unit}} \qquad \frac{\Gamma \vdash V_1\colon B_1 \quad \Gamma \vdash V_2\colon B_2}{\Gamma \vdash (V_1, V_2)\colon B_1 {*} B_2}$$

Finally, if we have a *closed* Freyd-category (Definition 2), we can interpret λ-terms (giving them a call-by-value semantics). We add multiplicative sequents for λ-abstraction and application:

$$\frac{\Gamma, x\colon A \vdash M\colon B}{\Gamma \vdash \lambda x.M\colon A \to B} \qquad \frac{\Gamma \vdash V\colon A \to B \quad \Delta \vdash W\colon A}{\Gamma, \Delta \vdash VW\colon B}$$

The adjunction giving the closed structure codifies two principles of the call-by-value λ-calculus: λ-abstractions are values; and only values may be substituted for arguments.

The calculus, upon the addition of the predicates and rules due to Moggi, is equivalent to the λ_c-calculus [7]: we do not have types TX as they are equivalent to certain exponentials, while Moggi's constructors $[\_]$ and μ may be taken to be given by certain λ-terms and applications respectively. One can avoid π-terms by use of pattern matching.

3.2 κ-calculus and closed κ-categories

In Hasegawa's account [2], higher-order structure was added to the κ-calculus by a binder dual to the κ-binder. This does not seem to generalize to call-by-value. Instead, we add a thunking construct to the κ-calculus that corresponds to the closed structure on a κ-category.

Hasegawa originally compared his first-order function abstraction $\kappa x.M$ to `lambda` in early Lisp (i.e. lacking first-class functions). More recently, such a first-order binder has been found useful in compiling [1, 5]; it was called λ_s in [1]. Roughly speaking the reason is that a curried function $\lambda x_1 .\dots\lambda x_n.M$ would need to build n intermediate closures, an unnecessary expense if the function is immediately applied to n arguments. An uncurried n-ary function would avoid building closures, but not heap-allocation, as an n-tuple would have to be constructed before the function could be applied. By contrast, $\kappa x_1 .\dots\kappa x_n.M$ would simply consume n arguments by popping them from the stack. To turn this into a first-class function, a separate operation is needed (which we call `mkthunk`).

To adapt the κ-calculus to call-by-value, we introduce two kinds of judgements for values and computations, respectively. As usual, a value judgement is of the form $x_1 \colon C_1, \dots, x_n \colon C_n \vdash V \colon B$. A computation judgement of κ-calculus is of the form

$$x_1 \colon C_1, \dots, x_n \colon C_n \mid A_1, \dots, A_m \mapsto M \colon B_1, \dots, B_k$$

where A_i and B_j are types. Intuitively, $A_1, \dots, A_m$ is the type of the stack *before* M is run, and $B_1, \dots, B_k$ the type of the stack *after* M has run. (Douence and Fradet [1] also proposed a type system for their calculus, but typing failed to be preserved by the associativity law for sequential composition, essentially because their types do not keep track of the whole stack.) Our version of κ-calculus is a combination of [1] and [2]:

$$\frac{\Gamma \mid S \mapsto M \colon S' \quad \Gamma \mid S' \mapsto N \colon S''}{\Gamma \mid S \mapsto M; N \colon S''} \qquad \frac{}{\Gamma, x \colon A, \Gamma' \vdash x \colon A}$$

$$\frac{\Gamma \vdash V \colon C}{\Gamma \mid S \mapsto \mathtt{push}\, V \colon C, S} \qquad \frac{\Gamma, x \colon C \mid S \mapsto M \colon S'}{\Gamma \mid C, S \mapsto \kappa x.M \colon S'}$$

$$\frac{\Gamma \mid S \mapsto M \colon S'}{\Gamma \vdash \mathtt{mkthunk}\, M \colon [S \to S']} \qquad \frac{}{\Gamma \mid [S \to S'], S, S'' \mapsto \mathtt{apply} \colon S', S''}$$

A computation judgement denotes a morphism from $A_1 \times \cdots \times A_m$ to $B_1 \times \cdots \times B_k$ in the fibre over $C_1 \times \cdots \times C_n$. A value judgement denotes a morphism in the base in the usual fashion. Most of the categorical semantics is fairly evident from the syntax and follows Hasegawa [2]; for instance ";" is interpreted by composition (in diagrammatic order) in the fibre over $[\![\Gamma]\!]$. The κ-binder is interpreted by the κ-adjunction (Definition 6). The denotation of $\mathtt{push}\,V$ is given by reindexing $\kappa^{-1}\mathsf{id}_{C\times S}$ along the denotation of V. The higher-order structure in terms of `mkthunk` is given by the adjunction making a κ-category closed (Definition 3). (A similar decomposition of λ into binding and thunking underlies Levy's call-by-push-value paradigm [6].)

The intended meaning of the κ-calculus is that the most recently pushed value is popped by κ, and that `apply` forces a thunk (made by `mkthunk`) from the top of the stack:

$$\begin{aligned}(\mathtt{push}\,V);(\kappa x.M) &= M[x:=V]\\ (L;M);N &= L;(M;N)\\ \mathtt{push}\,(\mathtt{mkthunk}\,M);\mathtt{apply} &= M\end{aligned}$$

3.3 Syntactic translations

Because of the categorical equivalence results, we know that the languages in this section are in some sense equivalent. Here we give syntactic translations motivated by that equivalence.

In one direction, Douence and Fradet [1] remark that the `let` of the computational λ-calculus can be translated into the κ-binder. The translation of λ-calculus into κ-calculus is then essentially the same as their compilation in Figure 4 of [1]:

$$\begin{aligned}(\mathtt{let}\,x = M\,\mathtt{in}\,N)^\dagger &= M^\dagger;\kappa x.N^\dagger\\ x^\dagger &= \mathtt{push}\,x\\ (\lambda x.M)^\dagger &= \mathtt{push}\,(\mathtt{mkthunk}\,(\kappa x.M^\dagger))\\ VW^\dagger &= W^\dagger;V^\dagger;\mathtt{apply}\end{aligned}$$

The translation of `let` explains the naming and sequencing of λ_c in terms of the pushing and popping of κ-calculus. For example, we read the right hand side of the equation as: evaluate M, leaving a value on the top of the stack; then pop that value and call it x in N.

It is slightly less obvious if κ-calculus can be translated into computational λ-calculus. After all, there is no stack for the κ-binder to pop. A solution is to provide a variable that acts as a stack. A κ-term M is translated into a λ-term M^* with a distinguished free variable s as follows:

$$\begin{aligned}(\kappa x.M)^* &= \mathtt{let}\,(x,s) = s\,\mathtt{in}\,M^*\\ (M;N)^* &= \mathtt{let}\,s = M^*\,\mathtt{in}\,N^*\\ x^* &= x\end{aligned}$$

$$\begin{aligned}(\texttt{push}\,V)^* &= (V^*, s)\\ (\texttt{mkthunk}\,M)^* &= \lambda s.M^*\\ \texttt{apply}^* &= \texttt{let}\,(f,x) = s\,\texttt{in}\,f\,x\end{aligned}$$

The rule for `apply` is somewhat awkward, as it only preserves typing if the remainder of the stack is empty. For the general case, one would need to reformat the stack before and after the function is applied, in effect by defining coherence isomorphisms.

4 Conclusions and directions for further work

Because of the equivalence of closed Freyd-categories and closed κ-categories to a strong monad with Kleisli exponentials, the known soundness and completeness results for the traditional interpretation of computational λ-calculus carry over in a routine fashion. (Soundness and completeness of a graphical notation similar to our variant of computational λ-calculus was shown by Jeffrey [3].)

We believe our equivalence expresses, at an abstract level, a semantic link between computational λ-calculus and stack-based intermediate languages used in compiling. It seems encouraging in this regard that the decomposition of λ into a first-order binder (which we called κ) and the creation of a first-class function (an operation we called thunking) arose both in compiling and from the notion of closed structure on a κ-category. Though much more remains to be done to flesh out this connection, it seems that relatively low-level constructs can have clean categorical semantics.

Acknowledgements Thanks to Peter O'Hearn and Paul Levy for discussions.

References

1. R. Douence and P. Fradet. A Taxonomy of Functional Language Implementations Part I : Call-by-Value , INRIA Research Report No 2783, 1995.
2. M. Hasegawa. Decomposing typed lambda calculus into a couple of categorical programming languages, *Proc. CTCS*, Lect. Notes in Computer Science 953 (1995).
3. A. Jeffrey. Premonoidal categories and a graphical view of programs. `http://www.cogs.susx.ac.uk/users/alanje/premon/paper-title.html` .
4. G.M. Kelly. *The basic concepts of enriched categories*. CUP (1982).
5. X. Leroy. The ZINC experiment : an economical implementation of the ML language. Technical Report RT-0117, INRIA, Institut National de Recherche en Informatique et en Automatique, 1990.
6. P. B. Levy. Call-by-push-value: a subsuming paradigm (extended abstract). In J.-Y Girard, editor, *Typed Lambda-Calculi and Applications*, Lecture Notes in Computer Science, April 1999.
7. E. Moggi. Computational Lambda calculus and Monads, *Proc. LICS 89*, IEEE Press (1989) 14-23.
8. E. Moggi. Notions of computation and monads, Information and Computation 93 (1991) 55-92.

9. A.J. Power. Premonoidal categories as categories with algebraic structure (submitted).
10. A.J. Power and E.P. Robinson. Premonoidal categories and notions of computation, *Proc. LDPL '96*, Math Structures in Computer Science.
11. A.J. Power and H. Thielecke. Environments, Continuation Semantics and Indexed Categories, Proc. Theoretical Aspects of Computer Science, Lecture Notes in Computer Science (1997) 391-414.

A Freyd-categories and κ-categories

We refer the reader to [10] for the definition of symmetric premonoidal category and associated structures.

Definition 4. *A* Freyd-category *consists of a category $\mathcal{C}$ with finite products, a symmetric premonoidal category $\mathcal{K}$, and an identity on objects strict symmetric premonoidal functor $J: \mathcal{C} \longrightarrow \mathcal{K}$.*

We will sometimes write $\mathcal{K}$ for a Freyd-category, as the rest of the structure is usually implicit.

Definition 5. *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 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 $\mathsf{s}(\mathcal{K}): \mathbf{Comon}(\mathcal{K})^{\mathrm{op}} \longrightarrow \mathcal{C}at$. Given a category $\mathcal{C}$ with finite products, 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 $\kappa(J): \mathcal{C}^{\mathrm{op}} \longrightarrow \mathcal{C}at$ given by $\mathsf{s}(\mathcal{K})$ composed with the functor induced by J from $\mathcal{C} \cong \mathbf{Comon}(\mathcal{C})$ to $\mathbf{Comon}(\mathcal{K})$.

Definition 6. *A* κ-category *consists of a small category $\mathcal{C}$ with finite products, together with an indexed category $H: \mathcal{C}^{\mathrm{op}} \longrightarrow \mathcal{C}at$ such that*

- *for each object A of $\mathcal{C}$, $\mathbf{Ob}\, H_A = \mathbf{Ob}\, \mathcal{C}$, and for each arrow $f: A \longrightarrow B$ in $\mathcal{C}$, the functor $H_f: H_B \longrightarrow H_A$ is the identity on objects*
- *for each projection $\pi: B \times A \longrightarrow B$ in $\mathcal{C}$, the functor H_π has a left adjoint L_B given on objects by $- \times A$*

– (the Beck-Chevalley condition) *for every arrow $f{:}\,B \longrightarrow B'$ in $\mathcal{C}$, the natural transformation from $L_B \circ H_{f\times\mathsf{id}_A}$ to $H_f \circ L_{B'}$ induced by the adjointness is the identity.*

$$\begin{tikzcd}
H_{B'\times A} \arrow[r, "L_{B'}"] \arrow[d, "H_{f\times\mathsf{id}_A}"'] \arrow[dr, phantom, "\Longrightarrow"] & H_{B'} \arrow[d, "H_f"] \\
H_{B\times A} \arrow[r, "L_B"] & H_B
\end{tikzcd}$$

Proposition 4. *Given a κ-category $H{:}\,\mathcal{C}^{\mathrm{op}} \longrightarrow \mathcal{C}at$, 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 under the adjunction 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.*

$$\begin{tikzcd}
\mathbf{1} & A \arrow[r, "f"] & B & \mathbf{1} \\
A \arrow[d, "\mathsf{inc}_{\mathbf{1}}(f)"'] & \mathbf{1} \arrow[d, "H_f(\iota_B)"', ""{name=L}] & \mathbf{1} \arrow[d, "\iota_B", ""'{name=R}] & B \arrow[d, "\mathsf{id}_B"] \\
B & B & B & B
\arrow[from=R, to=L, mapsto, "H_f"']
\end{tikzcd}$$

This plus naturality determines the rest of the structure.

Theorem 4. *Let $\mathcal{C}$ be a small category with finite products. Given a κ-category $H{:}\,\mathcal{C}^{\mathrm{op}} \longrightarrow \mathcal{C}at$, there is a small Freyd-category $J{:}\,\mathcal{C} \longrightarrow \mathcal{K}$, unique up to isomorphism, for which H is isomorphic to $\kappa(J)$. The correspondence extends to an equivalence between the 2-category of small Freyd-categories and that of κ-categories.*