Electronic Notes in Theoretical Computer Science 20 (1999)
URL: http://www.elsevier.nl/locate/entcs/volume20.html 49 pages

Direct models of the computational lambda-calculus

Carsten Führmann[1]

Division of Informatics
University of Edinburgh
UK

Abstract

We introduce direct categorical models for the computational lambda-calculus. Direct models correspond to the source level of a compiler whose target level corresponds to Moggi's monadic models. That compiler is a generalised call-by-value CPS-transform. We get our direct models by identifying the algebraic structure on the Kleisli category that arises from a monadic model. We show that direct models draw our attention to previously inconspicuous, but important, classes of programs (e.g. central, copyable, and discardable programs), and we'll analyse these classes exhaustively—at a general level, and for several specific computational effects. Moreover, we show that from each direct model K we can recover the monadic model from which K arises as the Kleisli category.

1 Introduction

1.1 Direct style and monadic style

This article is about *direct-style* categorical semantics of call-by-value programming languages. A programming language can be direct-style or not, and the semantics of a direct-style language can be direct-style or not. Our *direct-style language* is the language in which we write our programs. For call-by-value, this could be C or Java, Lisp or ML. In a more idealised setting, we could consider call-by-value PCF, or the call-by-value lambda-calculus. In this article, we'll work with Eugenio Moggi's computational lambda-calculus (and extensions thereof) to allow a clean theoretical treatment.

In contrast to call-by-name, reasoning about call-by-value programs doesn't allow replacing variables with arbitrary expressions, because terms can have computational effects. For example, if the effect is 'non-termination'

[1] Supported by the EU TMR Marie Curie Research Training Grant N^{o}ERBFMBICT982906

(which we'll also call 'divergence'), then the equation $(\lambda x.1)\,\Omega \equiv 1$, where Ω is a diverging program, is obviously false for call-by-value.

There may be a compiler for our direct-style language. We'll consider only a certain kind of compilers. One example is Appel's compiler for Standard ML of New Jersey (SML/NJ) [2]. It translates ML programs into terms of a 'CPS-language' (CPS stands for 'continuation-passing style'), which is in a certain sense simpler than ML, and good for code optimisation. Appel's compiler is a real-life case of a *CPS-transform*. If the direct-style language is the call-by-value lambda-calculus, then CPS-language is a fragment of the $\lambda_{\beta\eta}$-calculus. (This fragment is equivalent to Thielecke's *CPS-calculus*.) There, in contrast to the direct-style language, we *can* replace variables by arbitrary terms. The latter CPS-transform is a special case of a *monadic-style* transform (MS transform) of the call-by-value lambda-calculus into Moggi's 'Meta-language', which is a fragment of the $\lambda_{\beta\eta}$-calculus plus type- and term-constructors that describe a *strong monad*. (We'll see that transform later in this introduction.) From the monadic-style transform, we obtain the CPS-transform by instantiating the strong monad with a *continuations monad*.

1.2 The computational lambda-calculus

In this article, we use the computational lambda-calculus as an idealised direct-style call-by-value language. We make a tiny change in that we use different names for the type constructors: Instead of T, $\times$, and 1, we use L, $\otimes$, and I, respectively. (We'll keep T, $\times$, and 1 for the Meta-language, however.) This avoids confusing direct style with monadic style, which would be fatal. Figures 1–3 show the term formation rules.

The computational lambda-calculus has two kinds of judgements. First, equations $(\Gamma \vdash M \equiv N : A)$, where $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are derivable sequents. Second, 'existence' judgements $(\Gamma \vdash M \downarrow A)$ (where $(\Gamma \vdash M : A)$ is a derivable sequent), which imply that in every judgement we can substitute M for any variable x of type A. Existence judgements hold for terms of the forms x, $[M]$, $(\lambda x : A.M)$, $*$, $\langle x, y\rangle$, and $\pi_i(x)$.

The type constructor L and the two monadic term formations rules in Figure 3 don't occur explicitly in typical real-life call-by-value languages. As we shall see (Proposition 3.17), in models of the computational lambda-calculus, LX is naturally isomorphic to $(I \rightharpoonup X)$ such that $[M]$ corresponds to $(\lambda x : 1.M)$, where x is fresh, and $\mu(M)$ corresponds to $M*$. The main characteristic of the computational lambda-calculus is that we can't generally replace variables by arbitrary terms, but only by values. For a detailed description of the derivation rules for equations and value judgements, please see [10].

var	$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$
let	$\dfrac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash \mathit{let}\, x = M\, \mathit{in}\, N : B}$
$*$	$\Gamma \vdash * : I$
$\langle -, - \rangle$	$\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \otimes B}$
π_i	$\dfrac{\Gamma \vdash M : A_1 \otimes A_2}{\Gamma \vdash \pi_i(M) : A_i}$
constant $f : A \longrightarrow B$	$\dfrac{\Gamma \vdash M : A}{\Gamma \vdash f(M) : B}$

Fig. 1. "Basic" terms of the computational lambda-calculus

λ	$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \rightharpoonup B}$
app	$\dfrac{\Gamma \vdash M : A \rightharpoonup B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$

Fig. 2. Higher-order terms of the computational lambda-calculus

$[-]$	$\dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : LA}$
μ	$\dfrac{\Gamma \vdash M : LA}{\Gamma \vdash \mu(M) : A}$

Fig. 3. Monadic terms of the computational lambda-calculus

1.3 Categorical semantics

1.3.1 Monadic semantics

The target of the monadic-style transform is Moggi's Meta-language. It is a fragment of the $\lambda_{\beta\eta}$-calculus plus extra structure, and its categorical models are called λ_C-models [10]. A λ_C-model is a category C with finite products plus a strong monad T and T-exponentials—that is, for all objects A and B, an exponential $(TB)^A$ of TB by A. A strong monad is a monad together with a strength—that is, a natural isomorphism $A \times TB \longrightarrow T(A \times B)$ with certain equational properties (see [11]). Figures 4 and 5 show the Meta-language and its categorical semantics. A sequent $(\Gamma \vdash M : A)$ denotes a morphism $[\![\Gamma]\!] \longrightarrow [\![A]\!]$ in the λ_C-model.

The lambda-calculus part of the Meta-language (Figure 4) doesn't need the monad, except the functor T to describe that all the exponentials that we need are of the form $(TB)^A$. In Figure 5, which describes the monadic part, η is the monad's unit, and μ is the monad's multiplication. The operational idea is that a term of the form $[M]$ stands for the delayed evaluation of M. Given $[M]$, the interpreter does nothing—that is, it returns $[M]$. Given $(\mathit{let}\, x \Leftarrow M' \mathit{\ in}\ N)$, the interpreter evaluates M' until it gets a term of the form $[M]$, and then it evaluates M, binds the result to x, and proceeds with N.

Example 1.1 For a simple λ_C-model, let $C = \mathit{Set}$, and let T be the obvious monad such that, for each set A, we have $TA = \{\bot\} \cup \{\llcorner a \lrcorner : a \in A\}$. Then the Kleisli category C_T is isomorphic to the category of sets and partial functions.

In general, the monad T can be seen as a parameter that stands for a computational effect. In the above example, the effect is divergence, but there are also state monads, exceptions monads, continuations monads, and so on.

1.3.2 The missing direct semantics

Like for the Meta-language, Moggi's semantics of the computational lambda-calculus too uses a λ_C-model C. Here, a sequent $(\Gamma \vdash M : A)$ denotes a morphism $[\![\Gamma]\!] \longrightarrow T[\![A]\!]$ of C. This semantics is the same as first perform-

Rule	Syntax	Semantics
var	$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$	$= A_1 \times \cdots \times A_n \xrightarrow{\pi_i} A_i$
λ	$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.M) : A \to TB}$	$= \Gamma \times A \xrightarrow{f} TB$ $= \Gamma \xrightarrow{\lambda f} (TB)^A$
ev	$\dfrac{\begin{array}{c}\Gamma \vdash M : (TB)^A \\ \Gamma \vdash N : A\end{array}}{\Gamma \vdash MN : TB}$	$= \Gamma \xrightarrow{f} (TB)^A$ $= \Gamma \xrightarrow{g} A$ $= \Gamma \xrightarrow{\langle f,g \rangle} (TB)^A \times A \xrightarrow{ev} TB$
$*$	$\Gamma \vdash * : 1$	$= \Gamma \xrightarrow{!} 1$
$\langle -, - \rangle$	$\dfrac{\begin{array}{c}\Gamma \vdash M : A \\ \Gamma \vdash N : B\end{array}}{\Gamma \vdash \langle M, N \rangle : A \times B}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma \xrightarrow{g} B$ $= \Gamma \xrightarrow{\langle f,g \rangle} A \times B$
π_i	$\dfrac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \pi_i(M) : A_i}$	$= \Gamma \xrightarrow{f} A_1 \times A_2$ $= \Gamma \xrightarrow{f} A_1 \times A_2 \xrightarrow{\pi_i} A_i$
$f : A \longrightarrow B$	$\dfrac{\Gamma \vdash M : A}{\Gamma \vdash f(M) : B}$	$= \Gamma \xrightarrow{g} A$ $= \Gamma \xrightarrow{g} A \xrightarrow{f} B$

Fig. 4. The Meta-language and its categorical semantics: The lambda-calculus part

ing the monadic-style transform into the Meta-language and then using the semantics of the latter. The monadic-style transform sends a type A of the computational lambda-calculus to a type $A^\sharp$ of the Meta-language (Fig. 6), and it sends a sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : A)$ of the computational lambda-calculus to a sequent $\left(x_1 : A_1^\sharp, \ldots, x_n : A_n^\sharp \vdash M^\sharp : T\left(A^\sharp\right)\right)$ of

Rule	Syntax	Semantics
$[-]$	$\dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : A}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma \xrightarrow{f} A \xrightarrow{\eta} TA$
let	$\dfrac{\Gamma \vdash M : TA \qquad \Gamma, x : A \vdash N : TB}{\Gamma \vdash \mathit{let}\, x \Leftarrow M\, \mathit{in}\, N : TB}$	$= \Gamma \xrightarrow{f} TA$ $= \Gamma \times A \xrightarrow{g} TB$ $= \Gamma \xrightarrow{\langle id,f\rangle} \Gamma \times TA \xrightarrow{t} T(\Gamma \times A)$ $\xrightarrow{Tg} TTB \xrightarrow{\mu} TB$

Fig. 5. The Meta-language and its categorical semantics: The monadic part

$$\begin{aligned}
(A \otimes B)^\sharp &= A^\sharp \times B^\sharp \\
I^\sharp &= 1 \\
(A \rightharpoonup B)^\sharp &= \left(T\left(B^\sharp\right)\right)^{\left(A^\sharp\right)} \\
(LA)^\sharp &= T\left(A^\sharp\right)
\end{aligned}$$

Fig. 6. Monadic-style transform from the computational lambda-calculus into Moggi's Meta-language: Types

the Meta-language (Fig. 7). Because the monadic-style transform is a compiler as explained in Section 1.1, we can conclude that Moggi's semantics of the computational lambda-calculus essentially performs a compilation.

By a *direct semantics*, I mean a semantics that doesn't compile. So, is there a direct-style categorical semantics of the computational lambda-calculus? If so, what kind of categories do we need? And what is the semantic counterpart of the monadic-style transform? In other words, is there a solution for X and the two dashed arrows in figure 8?

In this article, we'll find X, and we'll show that X is the only reasonable solution. We'll call the new categories *direct λ_C-models*. They are *abstract Kleisli-categories* with extra structure. We get abstract Kleisli-categories by identifying the available algebraic structure on Kleisli categories. As we shall see, every monad induces an abstract Kleisli-category. What's more interest-

$$
\begin{aligned}
x^\sharp &= [x] \\
(\mathit{let}\, x = M \;\mathit{in}\; N)^\sharp &= \big(\mathit{let}\, x \Leftarrow M^\sharp \;\mathit{in}\; N^\sharp\big) \\
*^\sharp &= [*] \\
\langle M, N\rangle^\sharp &= \big(\mathit{let}\, x \Leftarrow M^\sharp \;\mathit{in}\; \mathit{let}\, y \Leftarrow N^\sharp \;\mathit{in}\; [\langle x, y\rangle]\big) \\
(\pi_i(M))^\sharp &= \big(\mathit{let}\, x \Leftarrow M^\sharp \;\mathit{in}\; [\pi_i(x)]\big) \\
(f(M))^\sharp &= \big(\mathit{let}\, x \Leftarrow M^\sharp \;\mathit{in}\; f(x)\big) \\
[M]^\sharp &= \big[M^\sharp\big] \\
(\mu(M))^\sharp &= \big(\mathit{let}\, x \Leftarrow M^\sharp \;\mathit{in}\; x\big) \\
(\lambda x : A.M)^\sharp &= \big[\lambda x : A^\sharp.\, \big(M^\sharp\big)\big] \\
(MN)^\sharp &= \big(\mathit{let}\, f \Leftarrow M^\sharp \;\mathit{in}\; \mathit{let}\, y \Leftarrow N^\sharp \;\mathit{in}\; (fy)\big)
\end{aligned}
$$

Fig. 7. Monadic-style transform from the computational lambda-calculus into Moggi's Meta-language: Terms

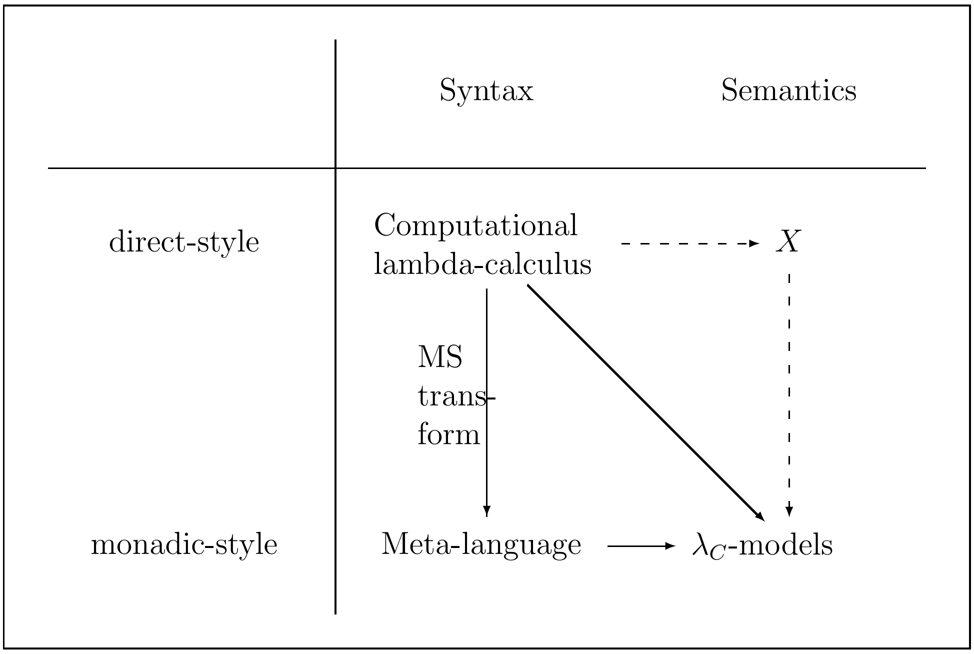


Fig. 8. The missing direct models, marked by X

ing is that from every abstract Kleisli-category K, we can recover a monad that induces K, up to isomorphism. Direct λ_C-models are to λ_C-models what abstract Kleisli-categories are to monads. Because Moggi's semantics of the computational lambda-calculus in a λ_C-model C uses only morphisms of the form $A \longrightarrow TB$, we can translate it into a direct semantics that uses direct λ_C-models. Because every λ_C-model induces a direct λ_C-model, and every direct λ_C-model arises from a λ_C-model, we neither loose nor gain generality by moving from λ_C-models to direct λ_C-models. We introduce direct λ_C-models in Section 2. In Section 3, we show that the computational lambda-calculus forms an internal theory for each direct λ_C-model. In Section 5, we analyse exhaustively the mathematical relation between λ_C-models and direct λ_C-models.

1.4 New insights from direct models

As it turns out, direct λ_C-models draw our attention towards previously inconspicuous classes of well-behaved morphisms: So-called thunkable morphisms, central morphisms, copyable morphisms, and discardable morphisms. As we shall see in Section 4, these classes are closed under composition (i.e.they form categories) and more, which means that good program properties propagate along the term structure. This seems to be important for justifying direct-style code optimisations. We'll also see in Section 4 that, for each computational effect, there is the fascinating mathematical challenge to find out how these classes are related.

Abstract Kleisli-categories are interesting from a purely mathematical point of view too. That every monad induces an abstract Kleisli-category, and every abstract Kleisli-category arises from a monad, follows from a stronger result: An evident category of abstract Kleisli-categories is reflective in an evident category of λ_C-models (Theorem 5.3). The analogous result holds for direct λ_C-models and λ_C-models (Theorem 5.18). The full subcategory of the category of monads that is equivalent under the reflection to the category of abstract Kleisli-categories is formed by the monads that fulfil Moggi's *equalizing requirement* (Theorem 5.23).

1.5 Related work

I found direct λ_C-models by analysing Hayo Thielecke's $\otimes\neg$-categories [17,18], which are direct models for call-by-value languages with higher-order types and continuations. (Roughly speaking, continuations bring the power of jumps into functional programming.) As shown in [4], $\otimes\neg$-categories are the direct λ_C-models that correspond to continuations monads.

Another approach to semantics of call-by-value languages is given by *Freyd categories* (called so by John Power—see [14,13]). A Freyd category comprises a given category C for modelling *values*, and a category K for modelling arbitrary terms. The main difference between Freyd categories and abstract Kleisli-categories is that the latter don't have a given category of values, but one that can be equationally defined.

One more approach is given by the *partially closed focal premonoidal categories* of Ralf Schweimeier and Alan Jeffrey [15]. They have *three* given categories: One for modelling values, one for modelling central expressions, and one for modelling arbitrary expressions. Jeffrey and Schweimeier use a nice graphical presentation of direct-style programs (see [8]).

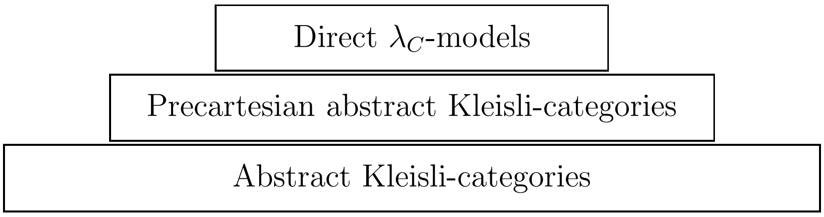
All approaches build on symmetric premonoidal categories, which were introduced by John Power and Stuart Anderson [1]. A gentle mathematical introduction to premonoidal categories, and an explanation of their relation to strong monads, is part of see [12]. However, I shall explain all necessary aspects of symmetric premonoidal categories in this article.

For direct semantics for call-by-value with recursion, we have *traced monoidal categories* [6,5]. They, however, rule out important premonoidal models, and therefore exclude many computational effects other than divergence, as Hasegawa pointed out in the conclusion of his thesis [6]. This limitation is broken by the *partially traced partially closed focal premonoidal categories* of Schweimeier and Jeffrey [15].

Anna Bucalo, Alex Simpson, and myself, in our article on equational lifting monads [3], use abstract Kleisli-categories as a vital tool to prove representation theorems for dominion

2 Direct models

In this section, we define the direct λ_C-models by giving structure in three layers:

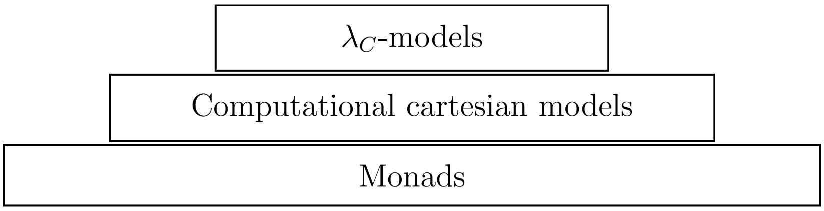


Direct λ_C-models

Precartesian abstract Kleisli-categories

Abstract Kleisli-categories

The hierarchy of direct models corresponds to the following hierarchy of monadic models:



λ_C-models

Computational cartesian models

Monads

In Section 2.1, we define abstract Kleisli-categories, and find the direct semantics for the monadic term formation rules of the computational lambda-calculus. In Section 2.2, we add structure to get precartesian abstract Kleisli-categories, and find the direct semantics for the basic term formation rules. In Section 2.3, we add structure to get direct λ_C-models, and find the direct semantics for the higher-order term formation rules.

2.1 Abstract Kleisli-categories

Suppose that T a monad with multiplication μ and unit η on a category C, and let C_T be the Kleisli category of T. As shown in [9], there is an adjunction

$$\begin{tikzcd} C \arrow[r, shift left=2, "F_T"] \arrow[r, phantom, "\perp"] & C_T \arrow[l, shift left=2, "G_T"] \end{tikzcd}$$

such that (1) $G_T F_T = T$, (2) the adjunction's unit is η, (3) letting ε be the counit, the $G_T \varepsilon F_T = \mu$, and (4) F_T is the identity on objects. The adjunction induces some structure on the Kleisli category: Let

$$L =_{\text{def}} F_T G_T$$
$$\vartheta =_{\text{def}} F_T \eta$$

We also have the counit ε, which is a natural isomorphism $L \longrightarrow Id$. For each object A, we have $\vartheta_A : A \longrightarrow LA$. However, ϑ is not generally a natural transformation $Id \longrightarrow L$, as we shall see. Of course, L, ϑ, and ε fulfil certain equations. This leads us to the following definition:

Definition 2.1 An *abstract Kleisli-category* is

- A category K
- A functor $L : K \longrightarrow K$
- A transformation[2] $\vartheta : Id \longrightarrow L$
- A natural transformation $\varepsilon : L \stackrel{\cdot}{\longrightarrow} Id$

such that ϑL is a natural transformation $L \stackrel{\cdot}{\longrightarrow} L^2$, and

$$\begin{tikzcd} Id \arrow[r, "\vartheta"] \arrow[d, "\vartheta"'] & L \arrow[d, "L\vartheta"] \\ L \arrow[r, "\vartheta L"] & L^2 \end{tikzcd} \qquad \begin{tikzcd} Id \arrow[r, "\vartheta"] \arrow[dr, "id"'] & L \arrow[d, "\varepsilon"] \\ & Id \end{tikzcd} \qquad \begin{tikzcd} L \arrow[r, "\vartheta L"] \arrow[dr, "id"'] & L^2 \arrow[d, "L\varepsilon"] \\ & L \end{tikzcd}$$

Let's pronounce ϑ 'thunk' and ε 'force', agreeing with Hayo Thielecke's terminology for $\otimes\neg$-categories, which turned out to be special abstract Kleisli-categories (see [4]).

Example 2.2 The well-known category *Rel* of sets and relations, which is isomorphic to the Kleisli category of the covariant powerset monad (see e.g. [7]). For a set A, let LA be the powerset of A. For a relation $R : A \longrightarrow B$, and

[2] By a transformation from a functor $F : C \longrightarrow D$ to a functor $G : C \longrightarrow D$, I mean a map that sends each object A of C to an arrow $FA \longrightarrow GA$

sets $X \subseteq A$ and $Y \subseteq B$, let (where $R[X]$ is the image of X under R)

$$\begin{aligned} X(LR)Y &\Leftrightarrow Y = R[X] \\ x\vartheta X &\Leftrightarrow X = \{x\} \\ X\varepsilon\, x &\Leftrightarrow x \in X \end{aligned}$$

Note that, for every abstract Kleisli-category, L forms a comonad on K with comultiplication ϑL and counit ε.

The next definition is crucial. Its importance will emerge gradually in this article.

Definition 2.3 A morphism $f : A \longrightarrow B$ of an abstract Kleisli-category K is called *thunkable* if

$$\begin{tikzcd} A \arrow[r, "\vartheta"] \arrow[d, "f"'] & LA \arrow[d, "Lf"] \\ B \arrow[r, "\vartheta"] & LB \end{tikzcd}$$

The category $G_\vartheta K$ is defined as the subcategory of K determined by all objects and the thunkable morphisms.

Example 2.4 In *Rel*, the thunkable morphisms are the total functions.

Note that, if K is the abstract Kleisli-category of a monad T, then a morphisms f of K is thunkable if and only if

$$f; \eta_T = f; T\eta$$

Now suppose that K is an arbitrary abstract Kleisli-category. For each $f \in K(A, B)$, let

$$[f] =_{\mathrm{def}} \vartheta; Lf$$

Letting *incl* be the inclusion functor $G_\vartheta K \hookrightarrow K$, there is an adjunction

$$[-] : K(\mathit{incl}(A), B) \cong (G_\vartheta K)(A, LB)$$

with unit ϑ and counit ε. Note that if K is the abstract Kleisli-category of a monad T, then

$$[f] = F_T f$$

Now suppose that C is a λ_C-model whose monad is $T = (T, \mu, \eta)$, and let K be the arising abstract Kleisli-category. Figures 9 and 10 show that we can use K to rewrite Moggi's monadic semantics of the monadic term formation rules and the existence judgement of the computational lambda-calculus. In [10], Moggi defines $(\Gamma \vdash M \downarrow A)$ to hold if the denotation f factors through η. In the presence of the equalizing requirement, which says that η_A is an equalizer

Rule	Syntax	Monadic semantics	Direct semantics
[−]	$\Gamma \vdash M : A$	$= \Gamma \xrightarrow{f} TA$	$= \Gamma \xrightarrow{f} A$
	$\Gamma \vdash [M] : LA$	$= \Gamma \xrightarrow{f} TA \xrightarrow{\eta} TTA$	$= \Gamma \xrightarrow{[f]} LA$
μ	$\Gamma \vdash M : LA$	$= \Gamma \xrightarrow{f} TTA$	$= \Gamma \xrightarrow{f} LA$
	$\Gamma \vdash \mu(M) : A$	$= \Gamma \xrightarrow{f} TTA \xrightarrow{\mu} TA$	$= \Gamma \xrightarrow{f} LA \xrightarrow{\varepsilon} A$

Fig. 9. Monadic semantics and direct semantics of the computational lambda-calculus' monadic term formation rules

Rule	Syntax	Monadic semantics	Direct semantics
ex	$\Gamma \vdash M : A$	$= \Gamma \xrightarrow{f} TA$	$= \Gamma \xrightarrow{f} A$
	$\Gamma \vdash M \downarrow A$	$\Leftrightarrow f; \eta_T = f; T\eta$	$\Leftrightarrow f$ is thunkable

Fig. 10. Monadic semantics and direct semantics of the computational lambda-calculus: The 'existence' judgement

of η_{TA} and $T\eta_A$, f factors through η if and only if $f; \eta_T = f; T\eta$. It turns out that this equation is the suitable interpretation of $(\Gamma \vdash M \downarrow A)$ in the absence of the equalizing requirement. This is so because in the computational lambda-calculus $(\Gamma \vdash M \downarrow A)$ is interderivable with the equation $(\Gamma \vdash \mathit{let}\, x = M\, \mathit{in}\, [x] \equiv [M] : LA)$ (see [10], bottom of p. 15), whose semantics is $f; \eta_T = f; T\eta$.

2.2 Precartesian abstract Kleisli-categories

A direct semantics of the computational lambda-calculus' basic term formation rules needs some kind of tensor for modelling product types, projections for modelling variables, and diagonals for modelling pairing. In this section, we're aiming for direct models that arise as Kleisli categories of Moggi's *computational cartesian models*. A computational cartesian model is a category C with given finite products and a strong monad T. Now let $t : A \times TB \longrightarrow T(A \times B)$ be the strength, and let $t' : (TA) \times B \longrightarrow T(A \times B)$ be the symmetric dual of t. The tensor and tensor unit are defined as in Figure 11. The diagonals,

$$\begin{aligned}
A \otimes B &=_{\text{def}} A \times B \\
I &=_{\text{def}} 1 \\
C \otimes f &=_{\text{def}} \Big(C \times A \xrightarrow{C \times f} C \times TB \xrightarrow{t} T(C \times B) \Big) \\
f \otimes C &=_{\text{def}} \Big(A \times C \xrightarrow{f \times C} B \times TC \xrightarrow{t'} T(B \times C) \Big)
\end{aligned}$$

Fig. 11. The premonoidal tensor and tensor unit on the Kleisli category of a cartesian computational model

projections, and structural isomorphisms of K are the images under F_T of the corresponding maps of C. This leads to the definition of *precartesian abstract Kleisli-categories.* I call them 'precartesian' because they are (symmetric) premonoidal categories together with projections and diagonals that don't quite form finite products. Roughly speaking, symmetric premonoidal categories are generalised symmetric monoidal categories in that the tensor doesn't have to be a bifunctor, but only a functor in either argument. Because not everybody knows symmetric premonoidal categories, I'll introduce them here (for more, see [12]). First, a couple of auxiliary definitions:

Definition 2.5 A *binoidal category* is

- A category C
- For each object A, a functor $A \otimes (-) : C \longrightarrow C$
- For each object B, a functor $(-) \otimes B : C \longrightarrow C$

such that for all objects A and B

$$(A \otimes (-))(B) = ((-) \otimes B)(A)$$

For the joint value, we write $A \otimes B$, or short AB.

Definition 2.6 A morphism $f : A \longrightarrow A'$ of a binoidal category K is called *central* if for each $g : B \longrightarrow B'$

$$\begin{tikzcd}
AB \arrow[r, "fB"] \arrow[d, "Ag"'] & A'B \arrow[d, "A'g"] & & BA \arrow[r, "Bf"] \arrow[d, "gA"'] & BA' \arrow[d, "gA'"] \\
AB' \arrow[r, "fB'"] & A'B' & & B'A \arrow[r, "B'f"] & B'A'
\end{tikzcd}$$

The *centre* ZK of K is defined as the subcategory determined by all objects and the central morphisms.

Definition 2.7 A *symmetric premonoidal category* is

- A binoidal category C

- An object I of C
- Four natural isomorphisms $A(BC) \cong (AB)C$, $IA \cong A$, $AI \cong A$, and $AB \cong BA$, with central components that fulfil the coherence conditions known from symmetric monoidal categories.

The symmetric monoidal categories are the symmetric premonoidal categories that have only central morphisms. As you can easily check, the natural associativity implies that $A \otimes (-)$ and $(-) \otimes A$ preserve central morphisms. Therefore

Proposition 2.8 *The centre of a symmetric premonoidal category is a symmetric monoidal category.*

Now for the definition of a precartesian abstract Kleisli-category. It is true, but not obvious, that every computational cartesian model induces a precartesian abstract Kleisli-category by the definitions in Figure 11. We'll show this later in this section.

Definition 2.9 A *precartesian abstract Kleisli-category K* is

- An abstract Kleisli-category K
- A symmetric premonoidal structure on K
- Finite products on $G_\vartheta K$ that agree with the symmetric premonoidal structure

such that $G_\vartheta K$ is a subcategory of the centre.

Example 2.10 We continue our example *Rel*. As we've seen in Example 2.2, *Rel* forms an abstract Kleisli-category whose thunkable morphisms are the total functions. The tensor on objects is the cartesian product of sets. On morphisms, the tensor is given by

$$(x, y)(R \otimes S)(x', y') \Leftrightarrow xRx' \wedge ySy'$$

As is well-known and obvious, the tensor forms a symmetric monoidal structure that agrees with the finite products on the subcategory of total functions. This example is a bit weak in that all morphisms are central, but we use it anyway in this section because it is short and elegant.

Now suppose that C is a computational cartesian model, and let K be the arising precartesian abstract Kleisli-category. Figure 12 shows how to use K to rewrite Moggi's monadic semantics of the computational lambda-calculus' basic term formation rules.

In the rest of this section, we'll show that every computational cartesian model induces a precartesian abstract Kleisli-category by the definitions in Figure 11. First we'll prove a proposition that helps checking whether a structure is a precartesian abstract Kleisli-category. Before stating the proposition, we define two important classes of morphisms: Discardable morphisms and copyable morphisms. For an object A of a precartesian abstract Kleisli-category K, let $!_A : A \longrightarrow I$ be the unique element of $(G_\vartheta K)(A, I)$, and let $\delta_A : A \longrightarrow AA$ be the obvious diagonal.

Definition 2.11 Suppose that K is a precartesian abstract Kleisli-category. A morphism $f \in K(A, B)$ is called *discardable* if

$$\begin{tikzcd}
A \arrow[r, "!"] \arrow[d, "f"'] & I \arrow[d, "id"] \\
B \arrow[r, "!"] & I
\end{tikzcd}$$

The category $G_! K$ is defined as the subcategory of K whose objects are those of K, and whose morphisms are the discardable morphisms of K.

Example 2.12 In *Rel*, a morphism $R : A \longrightarrow B$ is discardable if it relates every $x \in A$ with at least one element of B. So $G_! Rel$ is the same as the category Rel_{tot} in [7].

Next we define the notion of *copyable morphism*. In a precartesian abstract Kleisli-category, consider the two equations

$$\begin{tikzcd}
A \arrow[r, "\delta"] \arrow[dd, "f"'] & AA \arrow[d, "Af"] & & A \arrow[r, "\delta"] \arrow[dd, "f"'] & AA \arrow[d, "fA"] \\
 & AB \arrow[d, "fB"] & & & BA \arrow[d, "Bf"] \\
B \arrow[r, "\delta"] & BB & & B \arrow[r, "\delta"] & BB
\end{tikzcd}$$

Both have the same solutions f, because appending the twist map, which is absorbed by δ, transforms either equation into the other. But beware the conclusion that the two upper right paths are equal—they differ for continuations (proving this is outside the scope of this article).

Definition 2.13 A morphism $f : A \longrightarrow B$ of a precartesian abstract Kleisli-category K is called *copyable* if the two above equations hold. $G_\delta K$ is defined as the class of copyable morphisms of K.

In contrast to the discardable morphisms, the copyable ones don't always form a subcategory, as we'll see for global state (Example 4.7).

Example 2.14 In *Rel*, the copyable morphisms are the partial functions. So $G_\delta Rel$ is the same as the category Rel_{sval} in [7].

Proposition 2.15 *Suppose that K is an abstract Kleisli-category together with a binoidal tensor $\otimes$, an object I, and transformations $\delta_A : A \longrightarrow A \otimes A$, $\pi_i : A_1 \otimes A_2 \longrightarrow A_i$, and $!_A : A \longrightarrow I$. Then K determines a precartesian abstract Kleisli-category if and only if*

(i) *All morphisms of the form $[f]$ are central, copyable, and discardable.*

(ii) *All components of δ, π_i, and $!$, as well as all morphisms of the form $A \otimes [f]$ and $[f] \otimes A$, are thunkable.*

(iii) *Letting $\langle f, g\rangle =_{\mathrm{def}} \delta; id \otimes g; f \otimes id$, the transformations*

$$\begin{aligned}
\pi_1 &: A \otimes I \longrightarrow A \\
\pi_2 &: I \otimes A \longrightarrow A \\
\langle \pi_2, \pi_1\rangle &: A \otimes B \longrightarrow B \otimes A \\
\langle \pi_1;\pi_1, \langle \pi_1;\pi_2, \pi_2\rangle\rangle &: (A \otimes B) \otimes C \longrightarrow A \otimes (B \otimes C)
\end{aligned}$$

are natural in each argument.

(iv) *We have*

$$\begin{tikzcd}
AB \arrow[r, "\delta"] \arrow[dr, "id"'] & (AB)(AB) \arrow[d, "\pi_1 \otimes \pi_2"] \\
& AB
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "\delta"] \arrow[dr, "id"'] & A \otimes A \arrow[d, "\pi_i"] \\
& A
\end{tikzcd}$$

$$\begin{tikzcd}
A_1 \otimes A_2 \arrow[r, "\pi_1"] \arrow[d, "id \otimes !"'] & A_1 \\
A_1 \otimes I \arrow[ur, "\pi_1"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
A_1 \otimes A_2 \arrow[r, "\pi_2"] \arrow[d, "! \otimes id"'] & A_2 \\
I \otimes A_2 \arrow[ur, "\pi_2"'] &
\end{tikzcd}$$

Proof. As for the 'only if', all conditions hold because all morphisms of the form $[f]$ are thunkable and by definition the category of thunkable morphisms is a subcategory of the centre and has finite products.

Now for the 'if'. Because morphisms of the form $A \otimes [f]$ and $[f] \otimes A$ are thunkable, the thunkable morphisms are closed under $\otimes$. To see this, let $f \in (G_\vartheta K)(A, B)$. Then (recall that $\vartheta = [id]$)

$$\begin{aligned}
A \otimes f; \vartheta &= A \otimes f; \vartheta; L(A \otimes \vartheta); L(A \otimes \varepsilon) \\
&= A \otimes f; A \otimes \vartheta; \vartheta; L(A \otimes \varepsilon) \\
&= A \otimes [f]; \vartheta; L(A \otimes \varepsilon) \\
&= \vartheta; L(A \otimes [f]); L(A \otimes \varepsilon) \\
&= \vartheta; L(A \otimes f)
\end{aligned}$$

Because all morphisms of the form $[f]$ are central, so are all thunkable morphisms. To see this, let $f \in (G_\vartheta K)(A, B)$. Then for every $g \in K(A', B')$ we have

$$\begin{aligned} A \otimes g; f \otimes B' &= A \otimes g; [f] \otimes B'; \varepsilon \otimes B' \\ &= [f] \otimes A'; LB \otimes g; \varepsilon \otimes B' \\ &= f \otimes A'; \vartheta \otimes A'; LB \otimes g; \varepsilon \otimes B' \\ &= f \otimes A'; B \otimes g; \vartheta \otimes B'; \varepsilon \otimes B' \\ &= f \otimes A'; B \otimes g \end{aligned}$$

Because all morphisms of the form $[f]$ are copyable, so are all thunkable morphisms. To see this, let $f \in (G_\vartheta K)(A, B)$. Then

$$\begin{aligned} f; \delta &= f; \delta; \vartheta \otimes \vartheta; id \otimes \varepsilon; \varepsilon \otimes id \\ &= f; \vartheta; \delta; id \otimes \varepsilon; \varepsilon \otimes id \\ &= [f]; \delta; id \otimes \varepsilon; \varepsilon \otimes id \\ &= \delta; [f] \otimes [f]; id \otimes \varepsilon; \varepsilon \otimes id \\ &= \delta; [f] \otimes f; \varepsilon \otimes id \\ &= \delta; id \otimes f; f \otimes id \end{aligned}$$

Because all morphisms of the form $[f]$ are discardable, so are all thunkable morphisms. To see this, let $f \in (G_\vartheta K)(A, B)$. Then

$$f; ! = f; \vartheta; ! = [f]; ! = !$$

Now we can summarise that the thunkable morphisms determine a sub binoidal category of the centre that has only morphisms which are copyable and discardable, and contains δ, π_i, and !. Therefore the remaining conditions are enough to imply that $\otimes$, I, δ, π_i, and ! determine finite products on the category of thunkable morphisms. Every category with finite products determines a symmetric monoidal category (see [9], p. 159). Because the symmetric monoidal tensor of $G_\vartheta K$ agrees with the binoidal product on K, the symmetric monoidal structure on $G_\vartheta K$ extends to a symmetric premonoidal structure on K. □

Now we turn to prove a result which is slightly stronger than saying that the Kleisli category of a cartesian computational model forms a precartesian abstract Kleisli-category. This is a good time to recall two definitions from [12] and [14], respectively:

Definition 2.16 Suppose that C and D are symmetric premonoidal categories. Then a functor from C to D is *strict symmetric premonoidal* if it sends central morphisms to such and strictly preserves the tensor, the tensor unit, and the structural isomorphisms.

Definition 2.17 A *Freyd category* consists of a category C with finite products, a symmetric premonoidal category K with the same objects as C, and an identity-on-objects strict symmetric premonoidal functor $F : C \longrightarrow K$.

Proposition 2.18 *Suppose that C is a computational cartesian model. Then the functor $F_T : C \longrightarrow C_T$ together with the tensor and tensor unit defined in Figure 11 determines a Freyd category.*

Proof. In this proof, let's write semicolon for the composition of C_T, and colon for the composition of C. The structural premonoidal isomorphisms of C_T have to be the images of the evident corresponding maps of C. First we prove that, if C_T forms a symmetric premonoidal category, then F_T is strict symmetric premonoidal. The equation $F_T(A \times f) = A \otimes F_T f$ follows directly from the equation $id \times \eta : t = \eta$ in the definition of a computational cartesian model. Therefore, F_T strictly preserves the tensor. To see that all morphisms in the image of F_T are central, suppose that f is a morphism of C, and g is a morphism of C_T. Then

$$\begin{aligned}
(F_T f) \otimes id; id \otimes g &= F_T(f \times id); id \otimes g = f \times id : id \otimes g = f \times g : t \\
id \otimes g; (F_T f) \otimes id &= id \otimes g; F_T(f \times id) = id \otimes g : G_T F_T(f \times id) \\
&= id \times g : t : T(f \times id) = id \times g : f \times id : t
\end{aligned}$$

By definition, premonoidal structural isomorphisms are preserved by F_T.

Next we check that C_T forms a symmetric premonoidal category. That $A \otimes (-)$ preserves the composition of K follows from a routine calculation that uses the naturality of t and the equation in the definition of a computational cartesian model that involves t and μ. That $A \otimes (-)$ preserves the identity of K corresponds to the equation $id \times \eta : t = \eta$. By a symmetric argument, $(-) \otimes A$ too is a functor, so we have a binoidal category. The premonoidal structural isomorphisms are central because they are in the image of F_T, and their coherence follows from sending through F_T the coherence diagrams of the corresponding isos of C. So it remains to prove that the structural premonoidal isos of C_T are natural in each of their arguments. The naturality of the evident iso $I \otimes A \longrightarrow A$ follows from the naturality of the corresponding map π_2 of C and the equation $\left(1 \times TA \xrightarrow{t} T(1 \times A) \xrightarrow{T\pi_2} TA\right) = \pi_2$ in the definition of a computational cartesian model. The naturality of the symmetry map follows from the naturality of the corresponding map of C. The naturality of the associativity map follows from the naturality of the corresponding map of C, and the equation in the definition of a computational cartesian model that involves the strength and the associativity map of C. □

Proposition 2.19 *Suppose that $F : C \longrightarrow K$ is a Freyd category, and F has a right adjoint G with unit η and counit ε. Then K together with the $L =_{\mathrm{def}} GF$, $\vartheta =_{\mathrm{def}} F\eta$, and ε forms a precartesian abstract Kleisli-category, where δ, π_i, and $!$ are the images under F of the corresponding maps of C. Moreover, every morphism in the image of F is thunkable.*

Proof. All morphisms in the image of F are thunkable, because the equation $Fg; \vartheta = \vartheta; LFg$ is the image under F of the naturality square for η. No we use Proposition 2.15. By Proposition 2.18, $\otimes$ is in particular a binoidal tensor. For Condition (i) of Proposition 2.15, note that $[f] = F(f^\sharp)$, where $\sharp$ is the adjunction iso $K(FA, B) \cong C(A, GB)$. Therefore, $[f]$ is central. Because F preserves $\otimes$, δ, and $!$, all morphisms in the image of F are copyable and discardable, and therefore so is $[f]$. For Condition (ii), note that $A \otimes [f] = A \otimes F(f^\sharp) = F(A \times f^\sharp)$. Condition (iv) obviously holds, and Condition (iii) holds because the maps that must be natural are the given structural premonoidal isos of K. □

The proposition that we were aiming for follows directly from Propositions 2.18 and 2.19:

Proposition 2.20 *Suppose that C is a computational cartesian model. Then the Kleisli category C_T together with the tensor and tensor unit defined in Figure 11 forms a precartesian abstract Kleisli-category.*

2.3 Direct λ_C-models

In this section, finally, we shall define the direct models that arise as Kleisli categories of Moggi's λ_C-models. Suppose that C is a λ_C-model, and let K be the arising precartesian abstract Kleisli-category. We define higher-order structure on K as in Figure 13. This leads us to the definition of direct λ_C-models.

Definition 2.21 A *direct λ_C-model* is a precartesian abstract Kleisli-category K together with, for each object A, a functor $A \rightharpoonup (-) : K \longrightarrow G_\vartheta K$ and an adjunction

$$\Lambda : K(incl(B) \otimes A, C) \cong (G_\vartheta K)(B, A \rightharpoonup C)$$

Before we prove that the definitions in Figure 13 result in a direct λ_C-model, let's consider an example, and complete the direct semantics.

Example 2.22 We continue our example *Rel*. For sets A and B, let $A \rightharpoonup B$ be the set of relations $R \in A \times B$, and let

$$\begin{aligned} (R, x)\, apply\, y &\Leftrightarrow_{\text{def}} xRy \\ x(\Lambda R)S &\Leftrightarrow_{\text{def}} S = \{(y, z) : (x, y)Rz\} \end{aligned}$$

Now suppose that C is a λ_C-model, and let K be the arising direct λ_C-model. Figure 14 shows how to use K to rewrite Moggi's monadic semantics of the computational lambda-calculus' higher-order term formation rules.

In the rest of this section, we prove that the definitions in Figure 13 result in a direct λ_C-model. Like in the preceding section, we proceed in two steps. Now is a good time to recall a definition from [14].

Definition 2.23 A Freyd category $F : C \longrightarrow K$ is *closed* if for every objects A, the functor $(F-) \otimes A$ has a right adjoint.

Proposition 2.24 *If C is a λ_C-model, then the Freyd category $F_T : C \longrightarrow C_T$ is closed.*

Proof. In this proof, let's write semicolon for the composition of C_T, and colon for the composition of C. For each morphism $f \in C_T(C \otimes A, B)$ we need a unique solution $g \in C(C, A \rightharpoonup B)$ for the equation

$$\begin{tikzcd}
(A \rightharpoonup B) \otimes A \arrow[r, "apply"] & B \\
C \otimes A \arrow[u, "(F_T g) \otimes A"] \arrow[ur, "f"'] &
\end{tikzcd}$$

The required g is λf, where λ the Curry map $C(A \times B, TC) \cong C(A, (TC)^B)$, because $(F_T g) \otimes A; apply = F_T(g \times A); apply = g \times A : ev$. □

Proposition 2.25 *If $F : C \longrightarrow K$ is a closed Freyd-category, then K forms a direct λ_C-model.*

Proof. In this proof, let's write semicolon for the composition of C_T, and colon for the composition of C. Let $A \rightharpoonup -$ be the right adjoint of $(F-) \otimes A$, and *apply* the counit. For each morphism $f \in C(C \otimes A, B)$ we need a unique thunkable morphism $h : C \longrightarrow (A \rightharpoonup B)$ such that

$$\begin{tikzcd}
(A \rightharpoonup B) \otimes A \arrow[r, "apply"] & B \\
C \otimes A \arrow[u, "h \otimes A"] \arrow[ur, "f"'] &
\end{tikzcd}$$

Let λ be the adjunction isomorphism $K((FA) \otimes B) \cong C(A, B \rightharpoonup C)$. Because $F\lambda f$ is a solution for h, it remains to prove that every such solution is equal to $F\lambda f$. To see this, recall that saying h is thunkable is saying that $h : \eta_T = h : T\eta$, let $\sharp$ be the adjunction iso $K(FA, B) \cong C(A, GB)$, and consider

$$\begin{aligned}
F\lambda f &= F\lambda(h \otimes A; apply) = F\lambda([h] \otimes A; \varepsilon \otimes A; apply) \\
&= F\lambda(F(h^\sharp) \otimes A; \varepsilon \otimes A; apply) = F(h^\sharp : \lambda(\varepsilon \otimes A; apply)) \\
&= F(h^\sharp); F\lambda(\varepsilon \otimes A; apply) = [h]; F\lambda(\varepsilon \otimes A; apply) \\
&= h; \vartheta; F\lambda(\varepsilon \otimes A; apply) = h; F\eta; F\lambda(\varepsilon \otimes A; apply) \\
&= h; F(\eta : \lambda(\varepsilon \otimes A; apply)) = h; F(\lambda(F\eta \otimes A; \varepsilon \otimes A; apply)) \\
&= h; F(\lambda(\vartheta \otimes A; \varepsilon \otimes A; apply)) = h; F(\lambda apply) = h; Fid = h
\end{aligned}$$

□

The proposition that we were aiming for follows directly from Propositions 2.24 and 2.25:

Proposition 2.26 *Suppose that C is a λ_C-model. Then the precartesian abstract Kleisli-category C_T together with the definitions in Figure 13 forms a direct λ_C-model.*

3 Direct λ_C-models and the computational lambda-calculus

In this section, we translate *direct-style categorical expressions* like $A \otimes LB$ and $f; [g]$ into types and sequents, respectively, of the computational lambda-calculus. As we'll see, this translation is in a certain sense the inverse of the direct semantics. We'll use the inverse translation to prove soundness and completeness.

Definition 3.1 [Direct-style signature] For a collection $\mathcal{B}$ of base types, the *direct-style types over $\mathcal{B}$* are defined inductively by

$$A = LA \,|\, A \otimes A \,|\, I \,|\, A \rightharpoonup A \,|\, \mathcal{B}$$

A *direct-style constant over $\mathcal{B}$* is defined as a formal arrow $f : A \longrightarrow B$ where A and B are direct-style types over $\mathcal{B}$. A *direct-style signature* is defined as a pair $\Sigma = (\mathcal{B}, \mathcal{K})$ where $\mathcal{B}$ is a collection of base types, and $\mathcal{K}$ is a collection of direct-style constants over $\mathcal{B}$.

Next we define the domain of our inverse translation:

Definition 3.2 The *direct-style categorical expressions* over a direct-style signature $\Sigma = (\mathcal{B}, \mathcal{K})$ are defined inductively by

$$\begin{aligned} f =& id_A \,|\, f; f \,|\, A \otimes f \,|\, f \otimes A \,|\, \delta_A \,|\, \pi_1^{A,A} \,|\, \pi_2^{A,A} \,|\, !_A \\ & |\, [f] \,|\, \varepsilon_A \,|\, \Lambda f \,|\, \mathit{apply}_{A,A} \,|\, \mathcal{K} \end{aligned}$$

where A ranges over the direct-style types over $\mathcal{B}$, and we obey the obvious type constrains for semicolon and Λ.

Now for the range of the inverse translation:

Definition 3.3 The *computational lambda-sequents* over a direct-style signature $\Sigma = (\mathcal{B}, \mathcal{K})$ are defined as the sequents generated over the base types from $\mathcal{B}$ by the rules in Figures 1–3.

The direct semantics implicitly defines a syntactic translation from of computational lambda-sequents into direct-style categorical expressions. Let's write c for this syntactic version of the direct semantics. Now we turn to defining the inverse translation, which we call c'. For each direct-style signature Σ, c' takes direct-style categorical expressions over Σ to computational lambda-sequents over Σ. First some auxiliary definitions: For a direct-style type A over a collection of base types $\mathcal{B}$, the *factorisation* of A is defined as the sequence $A_1, \ldots, A_n$ of direct-style types over $\mathcal{B}$ such that none of the A_i is a product type or the unit type, and A is the product of the A_i up to associativity of the product and neutrality of the unit type. For example, the direct-style type

$$((B_0 \rightharpoonup B_1) \otimes LB_2) \otimes ((LB_1 \otimes (B_2 \rightharpoonup B_1)) \otimes I)$$

has the factorisation

$$B_0 \rightharpoonup B_1, LB_2, LB_1, B_2 \rightharpoonup B_1$$

The translation c' takes a direct-style categorical expression $f : A \longrightarrow B$ to a computational lambda-sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : B)$, where $A_1, \ldots, A_n$ is the factorisation of A. We use the following two abbreviations for direct-style lambda terms: First, suppose that $(\Gamma \vdash M : A)$ and $(y_1 : A_1, \ldots, y_n : A_n \vdash N : B)$ are computational lambda-sequents such that $A_1, \ldots, A_n$ is the factorisation of A. Then let

$$\begin{aligned}(\mathit{let}\, y_1, \ldots, y_n = M \,\mathit{in}\, N) =_{\text{def}} (&\mathit{let}\, z = M \,\mathit{in}\, \mathit{let}\, y_1 = p_1(z) \,\mathit{in}\, \ldots \\ &\mathit{let}\, y_n = p_n(z) \,\mathit{in}\, N)\end{aligned}$$

where z is a fresh variable of type A, and $p_i(z)$ is the obvious repeated application of π_1 and π_2 to z. Second, suppose that $(\Gamma, y_1 : A_1, \ldots, y_n : A_n \vdash M : B)$ is a computational lambda-sequent, and the direct-style type A has the factorisation $A_1, \ldots, A_n$. Then let

$$\lambda(y_1, \ldots, y_n) : A.M =_{\text{def}} (\lambda z : A.\mathit{let}\, y_1, \ldots, y_n = z \,\mathit{in}\, M)$$

where z is fresh. The translation c' proceeds by recursion over direct-style categorical expressions, as in figures 15–18.

The next two lemmas state that the translations c and c' are essentially inverse.

Lemma 3.4 *In every direct λ_C-model, for every direct-style combinator term m, letting i be the evident structural iso, we have*

$$c(c'(m)) = i; m$$

Proof. By induction over m. □

Lemma 3.5 *Suppose that Σ is a direct-style signature, and that $(\Gamma \vdash M : A)$ is a computational lambda-sequent over Σ such that, letting $(x_1 : A_1, \ldots, x_n : A_n) =_{\text{def}} \Gamma$, none of the A_i is a product type or the unit type. Suppose that M' is defined by*

$$(\Gamma \vdash M' : A) =_{\text{def}} c'(c(\Gamma \vdash M : A))$$

Then in the computational lambda-calculus over Σ *we have*

$$\Gamma \vdash M' \equiv M : A$$

Proof. By induction over $(\Gamma \vdash M : A)$. □

We could remove the restriction on the A_i in Lemma 3.5 by reorganising environments with respect to the tensor, but that doesn't seem to be worth the formal effort.

Definition 3.6 A *direct interpretation* of a direct-style signature $\Sigma = (\mathcal{B}, \mathcal{K})$ is a direct λ_C-model K together with, for each base type $A \in \mathcal{B}$, an object $[\![A]\!] \in Ob(K)$, and for each constant $(f : A \longrightarrow B) \in \mathcal{K}$, a morphism $[\![f]\!] \in K([\![A]\!], [\![B]\!])$.

Every direct λ_C-model K gives an obvious direct-style signature Σ_K, whose base types are the objects and whose constants are the morphisms. Trivially, K forms a direct interpretation of Σ_K. If we know K, then by 'computational lambda-sequents' we mean the ones over Σ_K.

Definition 3.7 A *computational lambda-theory* over a direct-style signature Σ is defined as a collection $\mathcal{T}$ of judgements $(\Gamma \vdash M \equiv N : A)$ and $(\Gamma \vdash M \downarrow A)$ where $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are computational lambda-sequents over Σ, such that $\mathcal{T}$ is closed under the deduction rules of the computational lambda-calculus.

Definition 3.8 A *direct model* of a computational lambda-theory $\mathcal{T}$ over a direct-style signature Σ is a direct-style interpretation of Σ that validates all judgements of $\mathcal{T}$.

Proposition 3.9 (Semantics of substitution) *In every direct* λ_C*-model* K*, for all computational lambda-sequents*

$$\Delta \vdash N : A$$
$$\Gamma, x : A, \Gamma' \vdash M : B$$

such that $(\Delta \vdash N \downarrow A)$*, we have*

$$[\![\Gamma, \Delta, \Gamma' \vdash M[x := N] : B]\!] = \left(\Gamma\Delta\Gamma' \xrightarrow{\Gamma[\![\Delta \vdash N : A]\!]\Gamma'} \Gamma A \Gamma' \xrightarrow{[\![\Gamma, x:A, \Gamma' \vdash M:B]\!]} B\right)$$

Proof. By induction over the sequent $(\Gamma, x : A, \Gamma' \vdash M : B)$. The case $M = x$ needs on obvious Lemma about the semantics of weakening, which is easy to prove. □

Like the two preceding lemmas, the next two propositions are somehow opposites of each other. Proposition 3.10 explains how to get a computational lambda-theory from a direct interpretation, and Proposition 3.12 goes in the opposite direction.

Proposition 3.10 (Soundness) *Suppose that K is a direct interpretation of a direct-style signature Σ. Then the judgements $(\Gamma \vdash M \equiv N : A)$ and $(\Gamma \vdash M \downarrow A)$ over Σ that hold in K form a computational lambda-theory.*

Proof. By checking that K validates each deduction rule of the computational lambda-calculus. That the true judgements are closed under substitution of variables by terms which 'exist' follows from Proposition 3.9. □

Definition 3.11 Suppose that $\mathcal{T}$ is a computational lambda-theory over a direct-style signature Σ. The binary relation $\approx_{\mathcal{T}}$ on the direct-style categorical expressions over Σ is defined as follows: Let $m \approx_{\mathcal{T}} n$ if and only if, letting $(\Gamma \vdash M : A) = c'(m)$ and $(\Gamma \vdash N : A) = c'(n)$, we have $(\Gamma \vdash M \equiv N : A)$ in $\mathcal{T}$.

Proposition 3.12 (Term model) *Suppose that $\mathcal{T}$ is a computational lambda-theory over a direct-style signature Σ. Then $\approx_{\mathcal{T}}$ is a congruence, and the congruence classes form a direct model of $\mathcal{T}$.*

Proof. The relation $\approx$ is a congruence because $\equiv$ is a congruence (That $\equiv$ is a congruence is stated explicitly by deduction rules of the computational lambda-calculus). So we have a graph whose nodes are the direct-style types over Σ, and whose arrows are $\approx$-classes of direct-style categorical expressions over Σ, and on this graph we have well-defined operators like composition of arrows, Λ, and so on. We check that we have a direct-style λ_C-model by working our way up through the structural layers: Category, abstract Kleisli-category, precartesian abstract Kleisli-category, direct λ_C-model. Because of Proposition 2.15, all we need to check is some equations. Each such equation e is checked by deriving in the computational lambda-calculus the equation which is the image of e under c'. The details are left as an exercise. Let's call our direct λ_C-model $K_{\mathcal{T}}$. Now it remains to prove that $K_{\mathcal{T}}$ is a direct model of $\mathcal{T}$. To see this, suppose that $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$, and let $m =_{\text{def}} c(\Gamma \vdash M : A)$ and $n =_{\text{def}} c(\Gamma \vdash N : A)$. We need to prove that $m \approx_{\mathcal{T}} n$. So let $(\Gamma \vdash M' : A) =_{\text{def}} c'(m)$ and $(\Gamma \vdash N' : A) =_{\text{def}} c'(n)$. We need $(\Gamma \vdash M' \equiv N' : A) \in \mathcal{T}$. This is so because, by Lemma 3.5, we have $(\Gamma \vdash M \equiv M' : A)$ and $(\Gamma \vdash N \equiv N' : A)$ in $\mathcal{T}$, and $\equiv$ is transitive. Now suppose that $(\Gamma \vdash M \downarrow A) \in \mathcal{T}$, and let $m =_{\text{def}} c(\Gamma \vdash M : A)$. We must prove that m is thunkable—that is $m; [id] \approx_{\mathcal{T}} [m]$. By definition of c', letting $(\Gamma \vdash M' : A) =_{\text{def}} c'(m)$, we must prove that $(\Gamma \vdash (\mathit{let}\, x = M' \,\mathit{in}\, [x]) \equiv [M'] : LA) \in \mathcal{T}$. Because $(\Gamma \vdash M \downarrow A) \in \mathcal{T}$, we have$(\Gamma \vdash (\mathit{let}\, x = M \,\mathit{in}\, [x]) \equiv [M] : LA) \in \mathcal{T}$. The rest follows because by Lemma 3.5, we have $(\Gamma \vdash M \equiv M' : A)$. □

Definition 3.13 For every computational lambda-theory $\mathcal{T}$, let $K_{\mathcal{T}}$ be the direct model induced by $\approx_{\mathcal{T}}$.

Proposition 3.14 (Completeness) *Suppose that $\mathcal{T}$ is a computational lambda-theory over a direct-style signature Σ, and that $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are computational lambda-sequents over Σ. If for every direct* *model K of $\mathcal{T}$ we have*

$$K[\![\Gamma \vdash M : A]\!] = K[\![\Gamma \vdash N : A]\!]$$

then

$$(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$$

If for every direct model K of $\mathcal{T}$ the morphism $K[\![\Gamma \vdash M : A]\!]$ is thunkable, then $(\Gamma \vdash M \downarrow A) \in \mathcal{T}$.

Proof. First let's prove the claim for the equation judgements. By hypothesis, we have $K_{\mathcal{T}}[\![\Gamma \vdash M : A]\!] = K_{\mathcal{T}}[\![\Gamma \vdash N : A]\!]$. By definition of $K_{\mathcal{T}}$, letting $m =_{\mathrm{def}} c(\Gamma \vdash M : A)$ and $n =_{\mathrm{def}} c(\Gamma \vdash N : A)$, we have $m \approx_{\mathcal{T}} n$. By definition of $\approx_{\mathcal{T}}$, letting $(\Gamma \vdash M' : A) =_{\mathrm{def}} c'(m)$ and $(\Gamma \vdash N' : A) =_{\mathrm{def}} c'(n)$, we have $(\Gamma \vdash M' \equiv N' : A) \in \mathcal{T}$. By Lemma 3.5, we have $(\Gamma \vdash M \equiv M' : A)$ and $(\Gamma \vdash N \equiv N' : A)$ in $\mathcal{T}$. Transitivity of $\equiv$ implies $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$. Proving the claim for the existence judgements is similar. □

Definition 3.15 Suppose that K and K' are direct interpretations of a direct-style signature Σ. Then a *morphism of direct interpretations of* Σ from K to K' is a morphism of direct λ_C-models from K to K' that strictly preserves all base types and constants of Σ.

Proposition 3.16 (Initiality) *Suppose $\mathcal{T}$ that is a computational lambda-theory over a direct-style signature Σ. Then $K_{\mathcal{T}}$ is an initial object in the category of direct models of $\mathcal{T}$ and morphisms of direct interpretations of Σ.*

Proof. Suppose that K is a direct model of $\mathcal{T}$, and that $H : K_{\mathcal{T}} \longrightarrow K$ is a morphism of direct interpretations of Σ. Then for each direct-style categorical expression m over Σ, because $K_{\mathcal{T}}[\![m]\!] = [m]_{\approx_{\mathcal{T}}}$, we must have $H([m]_{\approx_{\mathcal{T}}}) = K[\![m]\!]$. So it remains to prove that H is well defined. To see this, let $m \approx_{\mathcal{T}} n$. By definition of $\approx_{\mathcal{T}}$, letting $(\Gamma \vdash M : A) =_{\mathrm{def}} c'(m)$ and $(\Gamma \vdash N : A) =_{\mathrm{def}} c'(n)$, we have $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$. Soundness implies $K[\![\Gamma \vdash M : A]\!] = K[\![\Gamma \vdash N : A]\!]$, and therefore $c(c'(m)) = c(c'(n))$ holds in K. By Lemma 3.4, in K we have $c(c'(m)) = i; m$ and $c(c'(n)) = i; n$. By transitivity and cancelling i, we have $m = n$ in K. □

Proposition 3.17 *Suppose that K is a direct λ_C-model. Then there is a natural isomorphism*

$$LA \cong (I \rightharpoonup A)$$

that, for each $(\Gamma \vdash M : A)$, mediates by appending between $(\Gamma \vdash [M] : LA)$ and $(\Gamma \vdash \lambda x : I.M : I \rightharpoonup A)$, where x is fresh, and by prefixing between $(y : I \rightharpoonup A \vdash y* : A)$ and $(y' : LA \vdash \mu(y') : A)$.

Proof. The iso and its inverse are given by $(y : LA \vdash \lambda x : I.\mu(y) : I \rightharpoonup A)$, where x is fresh, and $(y' : I \rightharpoonup A \vdash [y'*] : LA)$. The required properties can be checked easily with the computational lambda-calculus. □

4 Varieties of morphisms

In this section, we'll examine some 'varieties of morphisms' of direct λ_C-models: thunkable morphisms, central morphisms, copyable morphisms, and discardable morphisms. To see what I mean by 'variety', let K be a direct λ_C-model, and let A and A' be objects of K. Then by *variety of morphisms of* $K(A, A')$, I mean a collection of elements of $K(A, A')$ that solve a collection of equations that use only the signature of direct λ_C-models. For example, the central morphisms $f : A \longrightarrow A'$ form a variety of $K(A, A')$, because they are the solutions of the following collection of equations:

$$\{fB; A'g = Ag; fB' : B, B' \in Ob(K), g \in K(B, B')\}$$

By a *variety of morphisms of* K, I mean a union over all objects A and A' of varieties of $K(A, A')$ that depend uniformly on A and A'. As we shall see, it is interesting to examine and relate these varieties—in general, and for specific computational effects.

4.1 The varieties in terms of the computational lambda-calculus

First, let's recap our knowledge about thunkable morphisms. By the direct semantics rule for existence predicates, in every direct λ_C-model, $(\Gamma \vdash M : A)$ is thunkable if and only if $(\Gamma \vdash M \downarrow A)$. By the deduction rules of the computational lambda-calculus, $(\Gamma \vdash M \downarrow A)$ implies that in all judgements we can substitute M for any free variable of type A. Moreover, $(\Gamma \vdash M \downarrow A)$ is interderivable with $(\Gamma \vdash (let\, x = M\, in\, [x]) \equiv [M] : LA)$. Therefore, $(\Gamma \vdash M : A)$ is thunkable if and only if in all judgements we can substitute M for any free variable of type A. Now let's look at the remaining three varieties.

Proposition 4.1 *In a direct λ_C-model K, for every computational lambda-sequent $(\Gamma \vdash M : A)$, the following are equivalent:*

(i) *The denotation of $(\Gamma \vdash M : A)$ is central.*

(ii) *For all sequents $(\Delta \vdash N : B)$ and $(\mathrm{E}, x : A, y : B \vdash O : C)$ where E contains all variables of Γ and Δ, we have*

$$\mathrm{E} \vdash (let\, x = M\, in\, let\, y = N\, in\, O) \equiv (let\, y = N\, in\, let\, x = M\, in\, O) : C$$

(iii) *For fresh variables $x : A$, $y : B$, and $z : LB$, we have*

$$\begin{aligned}\Gamma, z : LB \vdash &(let\, x = M\, in\, let\, y = \mu(z)\, in\, (x, y)) \\ \equiv &(let\, y = \mu(z)\, in\, let\, x = M\, in\, (x, y)) : A \otimes B\end{aligned}$$

Proof. Obviously, we have $(ii) \Rightarrow (iii)$. Now consider Condition (ii), and let

$$\begin{aligned}(f : \Gamma \longrightarrow A) &=_{\mathrm{def}} K[\![(\Gamma \vdash M : A)]\!]\\ (g : \Delta \longrightarrow B) &=_{\mathrm{def}} K[\![(\Delta \vdash N : B)]\!]\\ (h : \mathrm{E}AB \longrightarrow C) &=_{\mathrm{def}} K[\![\mathrm{E}, x : A, y : B \vdash O : C]\!]\end{aligned}$$

Let $d \in K(\mathrm{E}, \mathrm{E}\Gamma\Delta)$ be the obvious morphism given by the finite products on $G_\vartheta K$. Then the two morphisms

$$\begin{aligned}&K[\![\mathrm{E} \vdash (\mathit{let}\, x = M \;\mathit{in}\; \mathit{let}\, y = N \;\mathit{in}\, O) : C]\!]\\ &K[\![\mathrm{E} \vdash (\mathit{let}\, y = N \;\mathit{in}\; \mathit{let}\, x = M \;\mathit{in}\, O) : C]\!]\end{aligned}$$

are equal to, respectively,

$$\mathrm{E} \xrightarrow{d} \mathrm{E}\Gamma\Delta \xrightarrow{\mathrm{E}f\Delta} \mathrm{E}A\Delta \xrightarrow{\mathrm{E}Ag} \mathrm{E}AB \xrightarrow{h} C$$
$$\mathrm{E} \xrightarrow{d} \mathrm{E}\Gamma\Delta \xrightarrow{\mathrm{E}\Gamma g} \mathrm{E}\Gamma B \xrightarrow{\mathrm{E}fB} \mathrm{E}AB \xrightarrow{h} C$$

Obviously, the two agree if f is central, which proves $(i) \Rightarrow (ii)$. It remains to prove $(iii) \Rightarrow (i)$. In the case of Condition (iii), we get $g = \varepsilon_B$ and $h = \pi_2 : \mathrm{E}AB \longrightarrow AB$. So Condition (iii) implies

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

So by Proposition 5.8, f is central. □

Proposition 4.2 *In a direct λ_C-model, the denotation a sequent* $(\Gamma \vdash M : A)$ *is discardable if and only if for a fresh variable $x : A$, we have*

$$\Gamma \vdash (\mathit{let}\, x = M \;\mathit{in}\, *) \equiv * : I$$

Proof. Let f be the denotation of $(\Gamma \vdash M : A)$. By definition of the direct semantics, the right side of the equation denotes $!_\Gamma$. The left side denotes $(\delta; \Gamma f; !)$, and we have $\delta; \Gamma f; ! = \delta; \Gamma f; \pi_2; ! = \delta; \pi_2; f; ! = f; !$. □

Proposition 4.3 *In a direct λ_C-model, the denotation a sequent* $(\Gamma \vdash M : A)$ *is copyable if and only if for a fresh variable $x : A$, we have*

$$\Gamma \vdash (\mathit{let}\, x = M \;\mathit{in}\, \langle x, x\rangle) \equiv \langle M, M\rangle : A \otimes A$$

Proof. Let f be the denotation of $(\Gamma \vdash M : A)$. By definition of the direct semantics, the right side of the equation denotes $\delta; f\Gamma; Af$. The left side denotes $\delta; \Gamma f; \pi_2; \delta$, and we have $\delta; \Gamma f; \pi_2; \delta = \delta; \pi_2; f; \delta = f; \delta$. □

4.2 Closure properties

In this section, we determine for each of the four kinds of variety the operations under which it is closed in all models. For example, we'll see that the discardable morphisms are closed under identities, composition, $\otimes$, δ, π_i, and !, and that all morphisms of the form $[-]$ and λf are discardable. Such closure properties may help direct-style code optimisation. For example, the closure properties of discardable morphisms imply that discardable terms of the computational lambda-calculus are closed under all term formation rules except μ and *app*. So we can infer by induction over terms that a term M is discardable, and replace $(let\, x = M\; in *)$ with $*$.

First, let's recap the closure properties of thunkable morphism. As we know from Section 2.3, the thunkable morphisms form a computational cartesian model. In particular, they are closed under all operations of direct λ_C-models except (the nullary operations) ε and *apply*. Moreover, all morphisms of the form Λf and $[f]$ are thunkable.

Proposition 4.4 *The centre of a direct λ_C-model is closed under composition, identities, $\otimes$, δ, π_i, and !, and all morphisms of the forms Λf and $[f]$ are central. In particular, the centre is a symmetric monoidal category.*

Proof. Morphisms of the forms *id*, δ, π_i, !, Λf, and $[f]$ are thunkable and therefore central. Closure under composition is obvious. The closure under $\otimes$ follows from the naturality of the associativity iso. □

However, the centre doesn't generally have finite products. To see this, recall *Rel*, our leading example in Section 2.

Proposition 4.5 *The discardable morphisms of a direct λ_C-model are closed under composition, identities, $\otimes$, δ, π_i, !, and all morphisms of the forms Λf and $[f]$ are discardable. In particular, the discardable morphisms form a symmetric premonoidal category.*

Proof. The proof is analogous to that of Proposition 4.4, except for the tensor. To see the closure under the tensor, let f be a discardable morphism, and let A be an object. Then $Af; ! = Af; \pi_2; ! = \pi_2; f; ! = !$. □

The discardable morphisms don't generally form a symmetric monoidal category—continuations form a counterexample, which is outside the scope of this article.

Proposition 4.6 *The copyable morphisms of a direct λ_C-model are closed under identities, $\otimes$, δ, π_i, !, and all morphisms of the forms Λf and $[f]$ are copyable. But the copyable morphisms are not generally closed under composition.*

Proof. The proof of the closure properties is analogous to that of Proposition 4.4, except for the tensor. The closure under the tensor follows easily from the equation $\delta_{AB} = \delta_A \delta_B; i$ where i is the obvious structural iso. As shown in Example 4.7, the copyable morphisms of direct λ_C-models for global state aren't generally closed under composition. □

Example 4.7 [Global state] Suppose that C is a cartesian-closed category, and that S is some object of C. Using the internal language of C (i.e.the $\lambda_{\beta\eta}$-calculus), we define a direct λ_C-model like in Figure 19 (Note that an element f of $K(A, B)$ is a pair $\langle f_v, f_s\rangle$ that consists of a 'value component' $f_v \in C(A \times S, B)$ and a 'state component' $f_s \in C(A \times S, S)$). The direct λ_C-model K is isomorphic to the one that arises from the well-known monad $TX = (X \times S)^S$, which is also called the 'side-effects' monad. Formal element chasing shows that a morphism $f \in K(A, B)$ is copyable if and only if

$$f(a, s) = f(a, f_s(a, s))$$

For $A = B = I$ this means that f is an idempotent. And idempotents aren't generally closed under composition, as you can easily check for $C = Set$.

Proposition 4.8 *The morphisms of a direct λ_C-model which central and copyable are closed under composition, identities, $\otimes$, δ, π_i, and $!$. In particular, they form a symmetric monoidal category.*

Proof. We have all closure properties that hold for both central morphisms and copyable morphisms. It remains to show that the composition of two central, copyable morphisms is copyable, which is obvious. □

Proposition 4.9 *The morphisms of a direct λ_C-model which are central, copyable, and discardable are closed under composition, identities, $\otimes$, δ, π_i, $!$. Moreover, they form a category with finite products.*

Proof. An easy exercise. □

Problem 4.10 *Do the morphisms which are copyable and discardable form a category? (If so, it would have finite products.)*

We can think of the thunkable morphisms as effect free. By contrast, the components of ε have maximal effect, as the next proposition shows.

Proposition 4.11
- *If each component of ε is thunkable, then every morphism is thunkable.*
- *If each component of ε is central, then every morphism is central.*
- *If each component of ε is discardable, then every morphism is discardable.*
- *If each component of ε is copyable, then every morphism is copyable.*

Proof. First, note that every morphism f is equal to $[f]; \varepsilon$ and that $[f]$ is thunkable and therefore central, discardable, and copyable. The claims for thunkable, central, and discardable morphisms follow from closure under composition. For the claim about copyable morphisms, suppose that ε is copyable, and let f be any morphism. Then

$$\begin{aligned} f;\delta &= [f];\varepsilon;\delta = [f];\delta;id\otimes\varepsilon;\varepsilon\otimes id \\ &= \delta;id\otimes[f];[f]\otimes id;id\otimes\varepsilon;\varepsilon\otimes id \\ &= \delta;id\otimes[f];id\otimes\varepsilon;[f]\otimes id;\varepsilon\otimes id \\ &= \delta;id\otimes f;f\otimes id \end{aligned}$$

□

For some direct λ_C-models, the components of ε are in none of the four varieties:

Example 4.12 We continue Example 4.7. Let $f \in K(A, B) = C(A \times S, B \times S)$, and consider the following two equations

$$f_v(a, s) = f_v(a, s') \qquad (1)$$

$$f_s(a, s) = s \qquad (2)$$

Equation 1 means that the value of f doesn't depend on the store, and Equation 2 means that f doesn't write to the store. As follows from a routine calculation, f is thunkable if and only if both equations hold, and f is discardable if and only if Equation 2 holds. A slightly trickier argument shows that f is central if and only if f is thunkable. Note that discardable implies copyable. Obviously, the components of ε aren't generally copyable, and therefore they aren't generally in any of the other three varieties.

4.3 Relating the varieties

Figures 20 displays the four varieties (five, if we count the variety of all morphisms) for an arbitrary direct λ_C-model. In this section, we give enough examples to prove that all areas in the figure are inhabited in some model. Too see this, have Figures 21–26 handy. The areas $(ZK - G_!K) - G_\delta K$ and $(ZK \cap G_!K) - G_\delta K$ are obviously inhabited in *Rel*. The area $(ZK \cap G_\delta K) - G_!K$ too is inhabited in *Rel*, and for typical models of partiality. The area $(G_\delta K - G_!K) - ZK$ is inhabited for global state in the sense of Example 4.7, continuations and exceptions. The area $(G_!K - G_\delta K) - ZK$ is inhabited for continuations—proving this is outside the scope of this article. The area $(G_\delta K \cap G_!K) - ZK$ is inhabited for global state.

The only remaining area is $(ZK \cap G_!K \cap G_\delta K) - G_\vartheta K$. To see that it can be inhabited, consider the following example:

Example 4.13 Let C be a category with finite products. Let K be the category defined by

$$\begin{aligned} ObK &=_{\text{def}} ObC \\ K(A, B) &=_{\text{def}} C(A, B) \times C(A, B) \end{aligned}$$

with a component-wise definition of composition and identities. Let $A \otimes B =_{\text{def}} A \times B$ and $I =_{\text{def}} 1$, and define tensor, diagonal, and projections component-wise. Let

$$\begin{aligned} LA &=_{\text{def}} A \times A \\ L(f,g) &=_{\text{def}} (f \times g, f \times g) \\ \vartheta &=_{\text{def}} (\delta, \delta) \\ \varepsilon &=_{\text{def}} (\pi_1, \pi_2) \\ A \rightharpoonup B &=_{\text{def}} (B \times B)^A \quad (\text{or } B^A \times B^A) \end{aligned}$$

K forms a direct λ_C-model, which arises from an obvious λ_C-model with $TX = X \times X$. The tensor, tensor unit, diagonals, and projections form finite products on K. In particular, every morphism of K is discardable, copyable, and central. But the thunkable morphisms are those of the form (f, f).

5 Relating monadic models and direct models

As we've seen in Section 2, every monad induces an abstract Kleisli-category, and every λ_C-model induces even a direct λ_C-model, in which we can express the semantics of the computational lambda-calculus. In Section 3 we gave a direct-style proof of soundness and completeness. However, there is another way to get these results: The computational lambda-calculus is known to be sound and complete for λ_C-models (see [10]). So it must be complete for direct λ_C-models too, because if a judgement holds in all direct λ_C-models, then in particular it holds in those that arise from λ_C-models, and therefore it is derivable. For soundness of direct λ_C-models, it is enough to prove that each one such arises from a λ_C-model, because for each direct λ_C-model K, all derivable judgements hold in its generating λ_C-model, and therefore in K. In this section, we prove an even stronger result, which is also interesting as pure category theory.

5.1 Reflection and soundness

In this section, we define a suitable category of monads, and a suitable category of abstract Kleisli-categories, and prove that the second is reflective in the first. Then we'll build on this reflection to prove that precartesian abstract Kleisli-categories are reflective in computational cartesian models, and that direct λ_C-models are reflective in λ_C-models. Because the counit of a reflection is an isomorphism, we'll know that every direct λ_C-model is isomorphic to one that arises from a λ_C-model, and therefore we'll have soundness.

5.1.1 Abstract Kleisli-categories and monads

Definition 5.1 *AbsKl* is defined as the obvious category whose objects are abstract Kleisli-categories, and whose morphisms are functors that strictly preserve L, ϑ, and ε.

Definition 5.2 The category *Mnd* is defined as follows: An object is a pair $C = (C, T)$, where C is a category, and T is a monad on C. A morphism is a functor that strictly preserves the monad data.

If a C and D are categories, then let's write $C \lesssim D$ if C is reflective in D. (Because an adjunction is a reflection if and only if the right adjoint is full and faithful, $\lesssim$ is transitive. Because an adjunction is a reflection if and only if the counit is an iso, every equivalence of categories forms a reflection.)

Theorem 5.3 *The construction of the abstract Kleisli-category forms a functor Mnd* $\longrightarrow$ *AbsKl which is the left adjoint of a reflection*

$$\mathit{AbsKl} \lesssim \mathit{Mnd}$$

To prove the theorem, we define an intermediate category *KlAdj* of 'Kleisli adjunctions' such that $\mathit{AbsKl} \lesssim \mathit{KlAdj} \simeq \mathit{Mnd}$, and use transitivity.

Definition 5.4 A *Kleisli adjunction* is an adjunction $F \dashv G : K \longrightarrow C$ such that C and K have the same objects, and F is the identity on objects. The category *KlAdj* is defined as follows. Objects are Kleisli adjunctions. A morphism from $(F \dashv G : K \longrightarrow C)$ to $(F' \dashv G' : K' \longrightarrow C')$ is a pair of functors $h : C \longrightarrow C'$ and $H : K \longrightarrow K'$ such that h strictly preserves the unit, H strictly preserves the counit, and

$$\begin{tikzcd} C \arrow[r, "h"] \arrow[d, "F"'] & C' \arrow[d, "F'"] \\ K \arrow[r, "H"] & K' \end{tikzcd} \qquad \begin{tikzcd} C \arrow[r, "h"] & C' \\ K \arrow[r, "H"] \arrow[u, "G"] & K' \arrow[u, "G'"'] \end{tikzcd}$$

Lemma 5.5 *The construction of the Kleisli category forms an equivalence*

$$\mathit{Mnd} \simeq \mathit{KlAdj}$$

Proof. We define functors

$$\begin{tikzcd} \mathit{Mnd} \arrow[r, shift left, "\mathit{MakeKlAdj}"] & \mathit{KlAdj} \arrow[l, shift left, "\mathit{MakeMnd}"] \end{tikzcd}$$

and prove that they form an equivalence. The object part of *MakeKlAdj* is the well-known adjunction like in [9]:

$$\mathit{MakeKlAdj}(C, T) =_{\mathrm{def}} \left(\begin{tikzcd} C \arrow[r, shift left, "F_T"] \arrow[r, phantom, "\perp"] & C_T \arrow[l, shift left, "G_T"] \end{tikzcd} \right)$$

For $h \in \mathit{Mnd}(C, D)$ let

$$\mathit{MakeKlAdj}(h) =_{\mathrm{def}} (h, h)$$

That is, the component between the Kleisli categories acts just like h. Now for *MakeMnd*. For a Kleisli adjunction $F \dashv G : K \longrightarrow C$ of *KlAdj* with unit η and counit ε, let

$$\mathit{MakeMnd}(F \dashv G) =_{\mathrm{def}} (C, (GF, G\varepsilon F, \eta))$$

The morphism part of *MakeMnd* is obvious. We have $\mathit{MakeMnd} \circ \mathit{MakeKlAdj} = \mathit{Id}_{\mathit{Mnd}}$. So it remains to find a natural iso $\mathit{MakeKlAdj} \circ \mathit{MakeMnd} \cong \mathit{Id}_{\mathit{KlAdj}}$. Suppose that $F \dashv G : K \longrightarrow C$ is a Kleisli adjunction. We have the unique comparison functor $! : C_T \longrightarrow K$ (like in [9], page 144, theorem 2, where ! is called L). Let

$$E_{F \dashv G} =_{\mathrm{def}} (\mathit{Id}_C, ! : C_T \longrightarrow K) : \mathit{MakeKlAdj} \circ \mathit{MakeMnd} \cong \mathit{Id}_{\mathit{KlAdj}}$$

$E_{F \dashv G}$ is a morphism of Kleisli adjunctions from $F_T \dashv G_T$ to $F \dashv G$. Because F is the identity on objects, ! is an isomorphism (proving this is left as an exercise). Checking the naturality of E too is left as an exercise. □

Lemma 5.6 *There is a reflection*

$$\mathit{AbsKl} \lesssim \mathit{KlAdj}$$

Proof. We shall define a reflection

$$\begin{tikzcd} \mathit{AbsKl} \arrow[r, shift left=2, "\mathit{MakeKlAdj}"] \arrow[r, phantom, "\top"] & \mathit{KlAdj} \arrow[l, shift left=2, "\mathit{MakeAbsKl}"] \end{tikzcd}$$

For *MakeKlAdj*, let K be an abstract Kleisli-category. With unit ϑ and counit ε we have a Kleisli adjunction

$$\mathit{MakeKlAdj}(K) =_{\mathrm{def}} \left(\begin{tikzcd} G_\vartheta K \arrow[r, hook, shift left=2] \arrow[r, phantom, "\perp"] & K \arrow[l, shift left=2, "L"] \end{tikzcd} \right)$$

Now for the morphism part of *MakeKlAdj*. Suppose that $H : K \longrightarrow K'$ is a morphism of abstract Kleisli-categories. Because H strictly preserves L and ϑ, H preserves thunkable morphisms. So H has a restriction $H : G_\vartheta K \longrightarrow G_\vartheta K'$. Let

$$\mathit{MakeKlAdj}(H) =_{\mathrm{def}} (H : G_\vartheta K \longrightarrow G_\vartheta K', H : K \longrightarrow K')$$

Now for *MakeAbsKl*. For a Kleisli adjunction $F \dashv G : K \longrightarrow C$ with unit η and counit ε, let

$$\mathit{MakeAbsKl}(F \dashv G) =_{\text{def}} (K, L, \vartheta, \varepsilon)$$

where

$$\begin{aligned} L &=_{\text{def}} FG \\ \vartheta &=_{\text{def}} F\eta \end{aligned}$$

For the morphism part of *MakeAbsKl*, let

$$\mathit{MakeAbsKl}(h, H) =_{\text{def}} H$$

We have $\mathit{MakeAbsKl} \circ \mathit{MakeKlAdj} = \mathit{Id}_{\mathit{AbsKl}}$. Therefore we define the counit $\mathit{MakeAbsKl} \circ \mathit{MakeKlAdj} \longrightarrow \mathit{Id}_{\mathit{AbsKl}}$ of the reflection as the identity on $\mathit{Id}_{\mathit{AbsKl}}$. For the unit, which we'll call U, suppose that $F \dashv G : K \longrightarrow C$ is a Kleisli adjunction. If f is a morphism of C, then the square expressing that Ff is thunkable is the image of the square $f;\eta = \eta; GFf$ under F. So F has a corestriction to $G_\vartheta K$. Let

$$U_{F \dashv G} =_{\text{def}} (F : C \longrightarrow G_\vartheta K, \mathit{Id}_K)$$

Checking the naturality of U is left as an exercise. □

5.1.2 Precartesian abstract Kleisli-categories and cartesian computational models

In this section, we extend Theorem 5.3 to state that a suitable category of precartesian abstract Kleisli-categories is reflective in a suitable category of computational cartesian models.

Definition 5.7 *PrecAbsKl* is defined as the obvious category formed by the precartesian abstract Kleisli-categories and the functors that strictly preserve all operators (which are: L, ϑ, ε, $\otimes$, I, δ, π_i, and !).

Morphisms of *PrecAbsKl* would be strict symmetric premonoidal functors if they sent each central map to a central map. It isn't obvious that the images of central maps are central: For suppose that $F : K \longrightarrow K'$ is a morphism of precartesian abstract Kleisli-categories, and f is a central morphism of K. That f is central means that f commutes in the sense of Definition 2.6 with all morphisms g of K. Therefore, Ff commutes with all morphisms of the form Fg. But K' may have morphisms that are not in the image of F. Fortunately, we have

Proposition 5.8 *Suppose that K is a precartesian abstract Kleisli-category.*

A morphism $f \in K(A, A')$ is central if for all $B \in Ob(K)$

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

Proof. Let $g \in K(B, B')$. We have $g = [g]; \varepsilon$. Because $[g]$ is thunkable and therefore central, f commutes with $[g]$. Because f commutes with ε too, f commutes with g. □

Because morphisms of precartesian abstract Kleisli-categories preserve ε, we have

Corollary 5.9 *Morphisms of precartesian abstract Kleisli-categories preserve central morphisms.*

Definition 5.10 *Ccm* is defined as the obvious category formed by cartesian computational models and the morphisms of *Mnd* that strictly preserve the finite products and the strength.

Theorem 5.11 *The construction of the precartesian abstract Kleisli-category forms a functor $Ccm \longrightarrow PrecAbsKl$ which is the left-adjoint of a reflection*

$$PrecAbsKl \lesssim Ccm$$

To prove this, we define an intermediate category *IFreyd*, whose objects are Kleisli adjunctions with extra structure, such that $PrecAbsKl \lesssim IFreyd \simeq Ccm$, and then we use transitivity. First we define the objects of *IFreyd*.

Definition 5.12 A *I-closed Freyd-category* is a Freyd category $F : C \longrightarrow K$ together with a right adjoint of F.

I call them *I*-closed Freyd-categories because they are between Freyd categories and closed Freyd-categories: In a closed Freyd-category, for each object A, the functor $F(-) \otimes A$ has a right adjoint. In an *I*-closed Freyd-category, this needs to be so only for $A = I$.

Definition 5.13 The category *IFreyd* is defined as follows. The objects are the *I*-closed Freyd-categories. A morphism from $(F \dashv G : K \longrightarrow C)$ to $(F' \dashv G' : K' \longrightarrow C')$ is a morphism (h, H) of *KlAdj* such that h strictly preserves finite products, and H is strict symmetric premonoidal.

Lemma 5.14 *The construction of the Kleisli category forms an equivalence*

$$Ccm \simeq IFreyd$$

Proof. We extend the equivalence which is formed by *MakeKlAdj* and *MakeMnd* to an equivalence which is formed by functors

$$\begin{tikzcd} \mathit{Ccm} \arrow[r, shift left, "\mathit{MakeIFreyd}"] & \mathit{IFreyd} \arrow[l, shift left, "\mathit{MakeCcm}"] \end{tikzcd}$$

The object part of *MakeIFreyd* follows directly from Proposition 2.18, and checking the morphism part is straightforward. For *MakeCcm*, suppose that $F \dashv G : K \longrightarrow C$ is an *I*-closed Freyd-category with iso $\sharp : K(FA, B) \cong C(A, GB)$ and counit ε. The required strength is

$$t =_{\mathrm{def}} (A \otimes \varepsilon)^{\sharp}$$

Checking t is left as an exercise, as well as checking the morphism part of *MakeCcm*. To see that the two extended functors are inverse up to natural iso, it is enough to check that $\mathit{MakeCcm} \circ \mathit{MakeIFreyd} = \mathit{Id}_{\mathit{Ccm}}$ (i.e. $\mathit{MakeCcm} \circ \mathit{MakeIFreyd}$ doesn't change the finite products and the strength), and that each component of E is a morphism of I-closed Freyd categories (i.e. the unique comparison functor that is part of every component of E is strict symmetric premonoidal). These two to checks are left as an exercise. □

Lemma 5.15 *There is a reflection*

$$\mathit{PrecAbsKl} \lesssim \mathit{IFreyd}$$

Proof. We extend the reflection $\mathit{MakeAbsKl} \dashv \mathit{MakeKlAdj}$ to a reflection

$$\begin{tikzcd} \mathit{PrecAbsKl} \arrow[r, shift left=2, "\mathit{MakeIFreyd}"] \arrow[r, phantom, "\top"] & \mathit{IFreyd} \arrow[l, shift left=2, "\mathit{MakePrecAbsKl}"] \end{tikzcd}$$

The object part of *MakeIFreyd* is obvious, and Corollary 5.9 implies that *MakeIFreyd* sends morphisms of *PrecAbsKl* to morphisms of *IFreyd*. The object part of *MakePrecAbsKl* follows directly from Proposition 2.19. Checking the morphism part is left as an exercise. Because $\mathit{MakePrecAbsKl} \circ \mathit{MakeIFreyd}$ doesn't change the finite products, it is equal to $\mathit{Id}_{\mathit{PrecAbsKl}}$. Because each component of the unit U of the reflection $\mathit{AbsKl} \lesssim \mathit{KlAdj}$ preserves the finite products, U determines a unit for the required reflection. □

5.1.3 Direct λ_C-models

In this section, we extend Theorem 5.11 to state that a suitable category of direct λ_C-models is reflective in a suitable category of λ_C-models.

Definition 5.16 $D\lambda_C$ is defined as the obvious category formed by direct λ_C-models and the morphisms of precartesian abstract Kleisli-categories that strictly preserve the higher-order structure.

Definition 5.17 λ_C is defined as the obvious category formed by λ_C-models and the morphisms of computational cartesian models that strictly preserve the T-exponentials.

Theorem 5.18 *The construction of the direct λ_C-model forms a functor $\lambda_C \longrightarrow D\lambda_C$ which is the left adjoint of a reflection*

$$D\lambda_C \lesssim \lambda_C$$

To prove this, we define an intermediate category *ClFreyd*, whose objects are closed Freyd-categories, such that $D\lambda_C \lesssim \mathit{ClFreyd} \simeq \mathit{Ccm}$, and then we use transitivity.

Definition 5.19 *ClFreyd* is defined as the obvious category formed by closed Freyd categories and morphisms of *I*-closed Freyd-categories that strictly preserve the higher-order structure.

Theorem 5.20 *The construction of the Kleisli category forms an equivalence*

$$\lambda_C \simeq \mathit{ClFreyd}$$

Proof. We extend the equivalence which is formed by *MakeIFreyd* and *MakeCcm* to an equivalence which is formed by functors

$$\begin{tikzcd} \lambda_C \arrow[r, shift left, "\mathit{MakeClFreyd}"] & \mathit{ClFreyd} \arrow[l, shift left, "\mathit{Make}\lambda_C"] \end{tikzcd}$$

The object part of *MakeClFreyd* follows directly from Proposition 2.24, and checking the morphism part is straightforward.

For $\mathit{Make}\lambda_C$, suppose that $F \dashv G : K \longrightarrow C$ is a closed Freyd category with adjunction iso $\sharp : K(FA, B) \cong C(A, GB)$. We define the T-exponentials on C by

$$\begin{aligned} (TB)^A &=_{\mathrm{def}} A \rightharpoonup B \\ ev &=_{\mathrm{def}} \mathit{apply}^\sharp \end{aligned}$$

Because $\mathit{Make}\lambda_C \circ \mathit{MakeClFreyd}$ doesn't change the T-exponentials, it is the identity on λ_C. It remains to prove that the components of the natural iso $E : \mathit{MakeClFreyd} \circ \mathit{Make}\lambda_C \cong \mathit{Id}_{\mathit{IFreyd}}$ preserve the higher-order structure. This is left as an easy exercise. □

Theorem 5.21 *There is a reflection*

$$D\lambda_C \lesssim \mathit{ClFreyd}$$

Proof. We extend the reflection *MakePrecAbsKl* $\dashv$ *MakeIFreyd* to a reflection

$$\begin{tikzcd} D\lambda_C \arrow[r, shift left=2, "\mathit{MakeClFreyd}"] \arrow[r, phantom, "\top"] & \mathit{ClFreyd} \arrow[l, shift left=2, "\mathit{MakeD}\lambda_C"] \end{tikzcd}$$

MakeClFreyd is obvious, and $\mathit{MakeD}\lambda_C$ follows directly from Proposition 2.25. Obviously we have $\mathit{MakeD}\lambda_C \circ \mathit{MakeClFreyd} = \mathit{Id}_{D\lambda_C}$. Because the unit U of the reflection $\mathit{PrecAbsKl} \lesssim \mathit{IFreyd}$ strictly preserves the higher-order structure, U determines a unit for the required reflection. □

5.2 The equalizing requirement

Because of the reflection $\mathit{AbsKl} \lesssim \mathit{Mnd}$, abstract Kleisli-categories correspond to the full subcategory of *Mnd* that is determined by the objects (C, T) for which the reflection's unit is an iso. In this section, we identify that subcategory.

Definition 5.22 A monad T with unit η fulfils the *equalizing requirement* if, for each object A, η_A is an equalizer of η_{TA} and $T\eta_A$. The category Mnd_{eq} is defined as the full subcategory of *Mnd* determined by the objects (C, T) such that T fulfils the equalizing requirement.

Theorem 5.23 *There is an equivalence of categories*

$$\mathit{Mnd}_{eq} \simeq \mathit{AbsKl}$$

We prove this theorem, we use an subcategory KlAdj_{eq} of *KlAdj* such that $\mathit{Mnd}_{eq} \simeq \mathit{KlAdj}_{eq} \simeq \mathit{AbsKl}$.

Definition 5.24 KlAdj_{eq} is the full subcategory of *KlAdj* determined by the objects $F \dashv G$ such that, if η stands for the unit, then for each object A, η_A is an equalizer of η_{GFA} and $GF\eta_A$.

The next lemma follows directly from Lemma 5.5.

Lemma 5.25 *There is an equivalence of categories*

$$\mathit{Mnd}_{eq} \simeq \mathit{KlAdj}_{eq}$$

Lemma 5.26 *Suppose that $F \dashv G : K \longrightarrow C$ is a Kleisli adjunction with defining isomorphism $\sharp : K(FA, B) \cong C(A, GB)$. Then an element f of $K(A, B)$ is thunkable if and only if*

$$\begin{tikzcd} A \arrow[r, "f^\sharp"] \arrow[d, "f^\sharp"'] & GFB \arrow[d, "\eta"] \\ GFB \arrow[r, "GF\eta"] & GFGFB \end{tikzcd}$$

Proof. Applying the inverse of $\sharp$ to either path of the diagram yields

$$\begin{tikzcd}
A \arrow[r, "F\eta"] \arrow[d, "f"'] & FGA \arrow[d, "FGf"] \\
B \arrow[r, "F\eta"'] & FGB
\end{tikzcd}$$

Because $L = FG$ and $\vartheta = F\eta$, the square states that f is thunkable. □

Lemma 5.27 *An object $F \dashv G : K \longrightarrow C$ of KlAdj is in KlAdj_{eq} if and only if F is faithful and every thunkable morphism of K is in the image of F.*

Proof. Suppose that $F \dashv G : K \longrightarrow C$ is a Kleisli adjunction. For the 'only if', let $F \dashv G$ be an object of KlAdj_{eq}. Suppose that f is a thunkable morphism of K. By Lemma 5.26, we have $f^\sharp; \eta_{GFB} = f^\sharp; GF\eta_B$. Because η_B is an equalizer of η_{GFB} and $GF\eta_B$, there is a unique $g : A \longrightarrow B$ such that $g; \eta_B = f^\sharp$. The inverse $\flat$ of $\sharp$ sends the equation $g; \eta_B = f^\sharp$ to $Fg = f$. So every thunkable morphism is the image under F of exactly one morphism. Because all morphisms in the image of F are thunkable, F is faithful.

Now for the 'if'. Because η is natural, we have

$$\eta_B; \eta_{GFB} = \eta_B; GF\eta_B$$

for all objects B. Let $g \in C(A, GFB)$ such that

$$g; \eta_{GFB} = g; GF\eta_B$$

We need a unique $f \in C(A, B)$ such that $f; \eta = g$. The adjunction iso $\flat$ sends the equation $f; \eta = g$ to $Ff = g^\flat$. By Lemma 5.26, $g^\flat$ is thunkable. So there is exactly one solution f. □

Lemma 5.28 *The functor MakeAbsKl forms an equivalence*

$$\mathit{KlAdj}_{eq} \simeq \mathit{AbsKl}$$

Proof. Suppose that K is an abstract Kleisli-category. By Lemma 5.27, $\mathit{MakeKlAdj}(K)$ is an object of KlAdj_{eq}. It remains to prove that the reflection unit U restricts to an iso $\mathit{Id}_{\mathit{KlAdj}_{eq}} \cong \mathit{MakeKlAdj} \circ \mathit{MakeAbsKl}$. Let $F \dashv G : K \longrightarrow C$ be an object of KlAdj_{eq}. Then $U_{F\dashv G}$ is an iso, because by Lemma 5.27, $F : C \longrightarrow G_\vartheta K$ is an iso. □

Let Ccm_{eq}, $\lambda_{C\,eq}$, and $\mathit{ClFreyd}_{eq}$ be the full subcategories of Ccm, λ_C, and $\mathit{ClFreyd}$, respectively, that are determined by the objects that fulfil the equalizing requirement. Then Lemmas 5.28 and 5.25 imply

Corollary 5.29 *There are equivalences of categories*

$$\begin{gathered} \mathit{Ccm}_{eq} \simeq \mathit{IFreyd}_{eq} \simeq \mathit{PrecAbsKl} \\ \lambda_{C\,eq} \simeq \mathit{ClFreyd}_{eq} \simeq D\lambda_C \end{gathered}$$

Acknowledgement

I am indebted to Hayo Thielecke, whose ⊗¬-categories were the main inspiration for my analysis of direct models. Thanks a lot to John Power for explaining to me premonoidal categories and more, and commenting on my work. Thanks to Peter Selinger for many discussions. Thanks to Stuart Anderson for commenting on several versions of this article. Thanks to Peter O'Hearn for constructive comments. And a historical remark: Recently, Alex Simpson made me aware of his 1993 LFCS Lab-Lunch talk [16]. There he sketched what I call direct models. He had already found the essence of my reflection theorem for monads (Theorem 5.3). But there was no way to transfer this to strong monads, because premonoidal categories had not yet emerged. Finally, I'd like to thank Paul Taylor for his prooftree and diagram macros.

References

[1] S.O. Anderson and A.J. Power. A representable approach to finite nondeterminism. *Theoretical Computer Science*, 177:3–25, 1997.

[2] Andrew W. Appel. *Compiling with Continuations.* Cambridge University Press, 1992.

[3] Anna Bucalo, Carsten Führmann, and Alex Simpson. Equational lifting monads.
Available from http://www.dcs.ed.ac.uk/home/car/research.htm, June 1999. Submitted.

[4] Carsten Führmann. Relating two models of continuations.
Available from http://www.dcs.ed.ac.uk/home/car/research.htm, November 1998.

[5] M. Hasegawa. Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In *Proc. 6th International Conference on Category Theory and Computer Science (CTCS'95)*, volume 1210 of *LNCS*, Nancy, April 1997. Springer Verlag.

[6] Masahito Hasegawa. *Models of Sharing Graphs (A Categorical Semantics of Let and Letrec).* Distinguished Dissertation Series. Springer-Verlag, 1999.

[7] Bart Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69(1):73–106, 1994.

[8] Alan Jeffrey. Premonoidal categories and a graphical view of programs. Available from http://klee.cs.depaul.edu/premon/, 1998.

[9] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer-Verlag, 1971.

[10] E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Edinburgh Univ., Dept. of Comp. Sci., 1988.

[11] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[12] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, October 1997.

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

[14] John Power and Hayo Thielecke. Environments in Freyd categories and κ-categories. ICALP'99, to appear, 1999.

[15] Ralf Schweimeier and Alan Jeffrey. A categorical and graphical treatment of closure conversion. In *Electronic Notes in Theoretical Computer Science. Proceedings MFPS XV*, volume 20, New Orleans, 1999.

[16] Alex Simpson. Towards algebraic semantics of programming languages. Notes for a talk at the LFCS Lab Lunch, March 1993.

[17] Hayo Thielecke. *Categorical Structure of Continuation Passing Style.* PhD thesis, University of Edinburgh, 1997.

[18] Hayo Thielecke. Continuation semantics and self-adjointness. In *Proceedings MFPS XIII*, Electronic Notes in Theoretical Computer Science. Elsevier, 1997.

Rule	Syntax	Semantics
var	$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$	$= A_1 \cdots A_n \xrightarrow{\pi_i} A_i$
let	$\begin{array}{c} \Gamma \vdash M : A \\ \Gamma, x : A \vdash N : B \\ \hline \Gamma \vdash \mathit{let}\, x = M \,\mathit{in}\, N : B \end{array}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma A \xrightarrow{g} B$ $= \Gamma \xrightarrow{\delta} \Gamma\Gamma \xrightarrow{\Gamma f} \Gamma A \xrightarrow{g} B$
$*$	$\Gamma \vdash * : I$	$= \Gamma \xrightarrow{!} I$
$\langle -, - \rangle$	$\begin{array}{c} \Gamma \vdash M : A \\ \Gamma \vdash N : B \\ \hline \Gamma \vdash \langle M, N \rangle : A \otimes B \end{array}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma \xrightarrow{g} B$ $= \Gamma \xrightarrow{\delta} \Gamma\Gamma \xrightarrow{f\Gamma} A\Gamma \xrightarrow{Ag} AB$
π_i	$\begin{array}{c} \Gamma \vdash M : A_1 \otimes A_2 \\ \hline \Gamma \vdash \pi_i(M) : A_i \end{array}$	$= \Gamma \xrightarrow{f} A_1 A_2$ $= \Gamma \xrightarrow{f} A_1 A_2 \xrightarrow{\pi_i} A_i$
$f : A \longrightarrow B$	$\begin{array}{c} \Gamma \vdash M : A \\ \hline \Gamma \vdash f(M) : B \end{array}$	$= \Gamma \xrightarrow{g} A$ $= \Gamma \xrightarrow{g} A \xrightarrow{f} B$

Fig. 12. Direct semantics of the computational lambda-calculus: Basic structure

$$\begin{aligned} A \rightharpoonup B &=_{\text{def}} (TB)^A \\ (\mathit{apply} \in K((A \rightharpoonup B) \otimes A, B)) &=_{\text{def}} \big(\mathit{ev} \in C\,\big((TB)^A \times A, TB\big)\big) \\ \Lambda(f \in K(A \otimes B, C)) &=_{\text{def}} F_T(\lambda(f \in C(A \times B, TC)) \end{aligned}$$

Fig. 13. The higher-order structure on the precartesian abstract Kleisli-category of a λ_C-model

Rule	Syntax	Semantics
λ	$\Gamma, x : A \vdash M : B$	$= \Gamma \times A \xrightarrow{f} B$
	$\Gamma \vdash \lambda x : A.M : A \rightharpoonup B$	$= \Gamma \xrightarrow{\Lambda f} (A \rightharpoonup B)$
app	$\Gamma \vdash M : A \rightharpoonup B$	$= \Gamma \xrightarrow{f} (A \rightharpoonup B)$
	$\Gamma \vdash N : A$	$= \Gamma \xrightarrow{g} A$
	$\Gamma \vdash MN : B$	$= \Gamma \xrightarrow{\langle f,g \rangle} (A \rightharpoonup B)A \xrightarrow{\mathit{apply}} B$

Fig. 14. Direct semantics of the computational lambda-calculus: Higher-order structure

$$c'(f : A \longrightarrow B) \;\; = x_1 : A_1 \ldots, x_n : A_n \vdash f(\bar{x}) : A$$

if f is a constant

$$\begin{array}{cl} c'(f : A \longrightarrow B) & = \Gamma \vdash M : B \\ c'(g : B \longrightarrow C) & = y_1 : B_1, \ldots, y_n : B_n \vdash N : C \\ \cline{1-1} c'(f;g : A \longrightarrow C) & = \Gamma \vdash \mathit{let}\, y_1, \ldots, y_n = M \,\mathit{in}\, N : C \end{array}$$

$$c'(\mathit{id} : A \longrightarrow A) \;\; = x_1 : A_1 \ldots, x_n : A_n \vdash \bar{x} : A$$

Fig. 15. Translating direct-style categorical expressions into the computational lambda-calculus: Rules for composition and identity

$$\frac{c'(f : A \longrightarrow B)}{c'(C \otimes f : C \otimes A \longrightarrow C \otimes B)} \begin{array}{l} = \Gamma \vdash M : B \\ = y_1 : C_1, \ldots, y_n : C_n, \Gamma \vdash \langle \bar{y}, M \rangle : C \otimes B \end{array}$$

$$\frac{c'(f : A \longrightarrow B)}{c'(f \otimes C : A \otimes C \longrightarrow B \otimes C)} \begin{array}{l} = \Gamma \vdash M : B \\ = \Gamma, y_1 : C_1, \ldots, y_n : C_n \vdash \langle M, \bar{y} \rangle : B \otimes C \end{array}$$

$$c'(\delta : A \longrightarrow A \otimes A) = x_1 : A_1 \ldots, x_n : A_n \vdash \langle \bar{x}, \bar{x} \rangle : A \otimes A$$

$$c'(\pi_1 : A \otimes B \longrightarrow A) = x_1 : A_1 \ldots, x_n : A_n, y_1 : B_1, \ldots, y_n : B_m \vdash \bar{x} : A$$

$$c'(\pi_2 : A \otimes B \longrightarrow A) = x_1 : A_1 \ldots, x_n : A_n, y_1 : B_1, \ldots, y_n : B_m \vdash \bar{y} : B$$

$$c'(! : A \longrightarrow I) = x_1 : A_1 \ldots, x_n : A_n \vdash * : I$$

Fig. 16. Translating direct-style categorical expressions into the computational lambda-calculus: Rules for tensor and finite products

$$\frac{c'(f : A \longrightarrow B)}{c'([f] : A \longrightarrow LB)} \begin{array}{l} = \Gamma \vdash M : B \\ = \Gamma \vdash [M] : LB \end{array}$$

$$c'(\varepsilon : LA \longrightarrow A) = x : LA \vdash \mu(x) : A$$

Fig. 17. Translating direct-style categorical expressions into the computational lambda-calculus: Rules for $[-]$ and ε

$$\frac{c'(f : A \otimes B \longrightarrow C)}{c'(\Lambda f : A \longrightarrow (B \rightharpoonup C))} \quad \begin{array}{l} = \Gamma, y_1 : B_1, \ldots, y_n : B_m \vdash M : C \\ = \Gamma \vdash \lambda(y_1, \ldots, y_n) : B.M : B \rightharpoonup C \end{array}$$

$$c'(\mathit{apply} : (A \rightharpoonup B) \otimes A \longrightarrow B) = f : A \rightharpoonup B, x : A \vdash fx : B$$

Fig. 18. Translating direct-style categorical expressions into the computational lambda-calculus: Rules for higher-order structure

$$\begin{aligned}
\mathit{Ob}(K) &=_{\mathrm{def}} \mathit{Ob}(C) \\
K(A, B) &=_{\mathrm{def}} C(A \times S, B \times S) \\
A \otimes B &=_{\mathrm{def}} A \times B \\
(C \otimes f)((c, a), s) &=_{\mathrm{def}} ((c, f_v(a, s)), f_s(a, s)) \\
(f \otimes C)((a, c), s) &=_{\mathrm{def}} ((f_v(a, s), c), f_s(a, s)) \\
\delta(a, s) &=_{\mathrm{def}} ((a, a), s) \\
!(a, s) &=_{\mathrm{def}} (*, s) \\
\pi_i((a_1, a_2), s) &=_{\mathrm{def}} (a_i, s) \\
LA &=_{\mathrm{def}} (AS)^S \\
[f](a, s) &=_{\mathrm{def}} (\lambda s'.f(a, s'), s) \\
\varepsilon(f, s) &=_{\mathrm{def}} fs \\
A \rightharpoonup B &=_{\mathrm{def}} (BS)^{AS} \\
(\Lambda f)(a, s) &=_{\mathrm{def}} (\lambda(b, s').f((a, b), s'), s) \\
\mathit{apply}((f, a), s) &=_{\mathrm{def}} f(a, s)
\end{aligned}$$

Fig. 19. Building the direct λ_C-model for global state from a cartesian-closed category

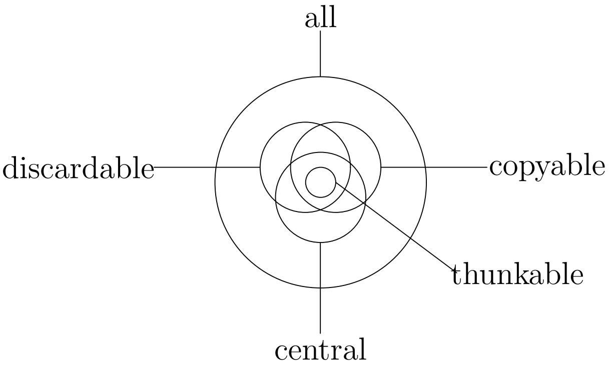


Fig. 20. Varieties of an arbitrary λ_C-model

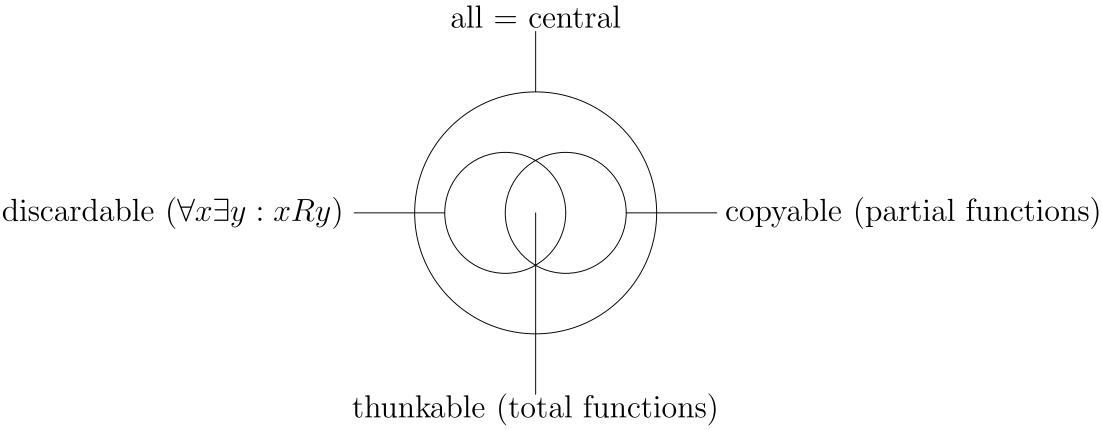

all = central
discardable ($\forall x \exists y : xRy$)
copyable (partial functions)
thunkable (total functions)

Fig. 21. Varieties of *Rel*

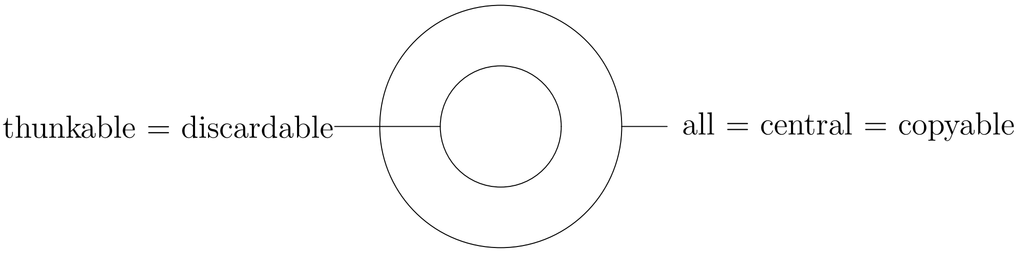


Fig. 22. Varieties of models of partiality

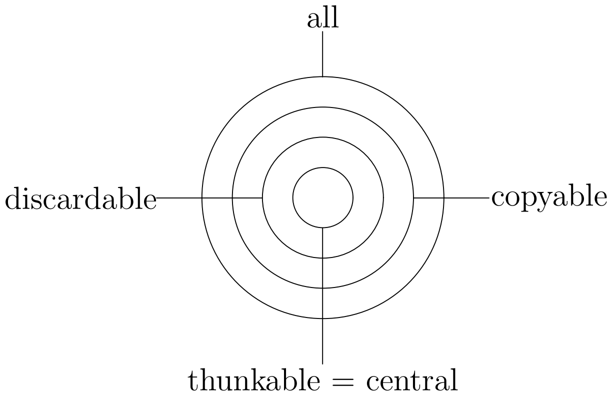


Fig. 23. Varieties for global state

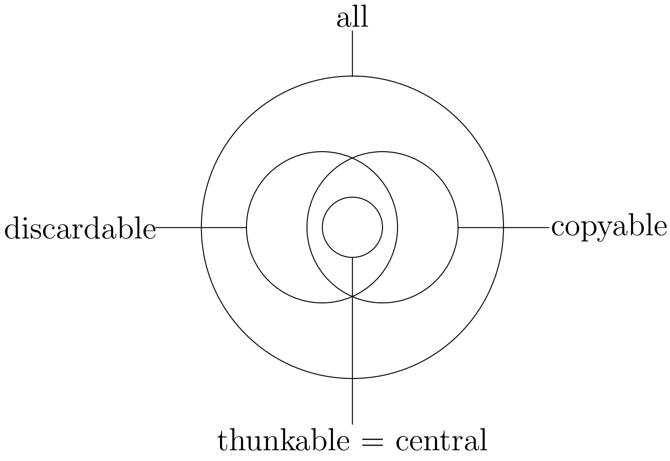


Fig. 24. Varieties of ⊗¬-categories (continuations)

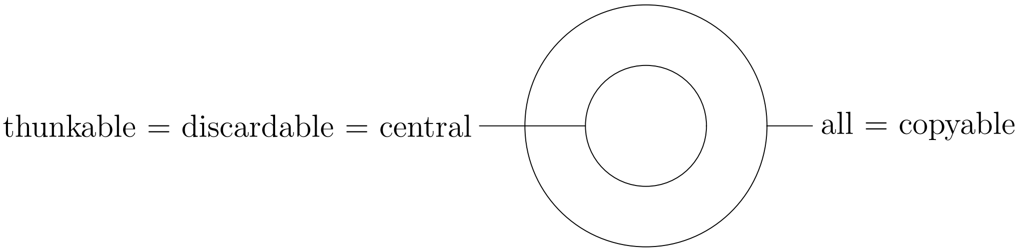


Fig. 25. Varieties for exceptions ($TX = X + E$)

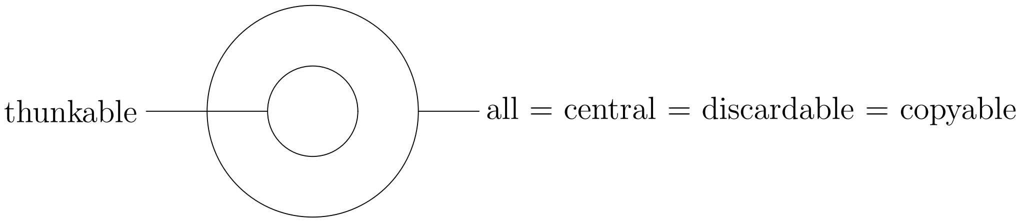


Fig. 26. Varieties for $TX = X \times X$