CTCS 2002 Preliminary Version

Adjunction Models For Call-By-Push-Value With Stacks

Paul Blain Levy

PPS, Université Denis Diderot, Case 7014, 2 Place Jussieu
75251 PARIS Cedex 05, France
Paul.Levy@pps.jussieu.fr

Abstract

Call-by-push-value (CBPV) is a new paradigm, which has been claimed to provide the semantic primitives from which call-by-value and call-by-name are built. We present its operational semantics in the form of a Felleisen-Friedman style CK-machine, and see how this machine suggests a new term judgement of stacks. When augmented with this judgement, CBPV has an elegant categorical semantics based on adjunctions.

We describe this categorical semantics incrementally. First, we introduce locally indexed categories and the opGrothendieck construction, and use these to give the basic structure for interpreting the 3 judgements: values, stacks and computations. Then we look at the universal property required to interpret each type constructor. We define a model to be a strong adjunction with countable coproducts, countable products and exponentials.

We justify this definition in two ways. First, we see that it has a wide range of instances: we give examples for divergence, storage, erratic choice, continuations etc., in each case decomposing Moggi's strong monad into a strong adjunction.

For the second justification, we start by giving equational laws for CBPV+stacks. This requires some additional pattern-matching constructs, but they do not affect the set of computations. We then show that the categories of theories and of models are equivalent.

1 Introduction

1.1 Background

Moggi [17] introduced the use of a *strong monad* T on a cartesian category $\mathcal{C}$ to model call-by-value languages. As noted by [5] and others, this structure can also be used to interpret call-by-name, where a type denotes a T-algebra.

Based on these ideas, the *call-by-push-value* (CBPV) paradigm was introduced, subsuming call-by-value and call-by-name. Two key type constructors

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

in CBPV are U and F, and their composite UF corresponds to Moggi's type constructor T. This immediately prompts the question: surely CBPV decomposes Moggi's monad into an adjunction? Now, this is certainly the case for all of the concrete models studied. For example the storage model decomposes Moggi's $S \to (S \times -)$ monad into $S \to -$ and $S \times -$, whilst the continuations model decomposes Moggi's $(- \to R) \to R$ monad into $- \to R$ and $- \to R$.

However, the syntax of CBPV does not confirm this analysis. CBPV has two judgements, *values* and *computations*, and the former give us a value category $\mathcal{C}$, but the other category required for an adjunction is absent. Thus, we are left with a "not-quite-adjunction" which can be formulated in several ways, none of them elegant [15].

But fortunately, recent work on CK-machine semantics (a form of operational semantics [4]) for CBPV has brought to light a new judgement: that of *stacks*. (Independently, a stack judgement with some similar rules was introduced in [3], in the setting of call-by-name and call-by-value with control effects.) The categorical semantics of CBPV+stacks is precisely the elegant adjunction structure noticed in each of the concrete models. The purpose of this paper is to present this adjunction semantics.

1.2 Adjunctions: A Discussion

Let $\mathcal{C}$ and $\mathcal{D}$ be categories; we will underline objects of $\mathcal{D}$. It is well known that the notion of *adjunction* from $\mathcal{C}$ to $\mathcal{D}$ has numerous equivalent definitions. One of these requires functors U and F and an isomorphism

$$\mathcal{C}(X, U\underline{Y}) \cong \mathcal{D}(FX, \underline{Y}) \quad \text{natural in } X \text{ and } \underline{Y} \tag{1}$$

Alternatively F can be specified on objects only and naturality in X removed. (This is equivalent to the first definition by the parametrized representability theorem.) But can we give a definition where both U and F are specified on objects only? Here is one way.

Let us say, in an adjunction, that an *oblique morphism* from X to $\underline{Y}$ is a $\mathcal{C}$-morphism from X to $U\underline{Y}$ or a $\mathcal{D}$-morphism from FX to $\underline{Y}$; it hardly matters which, since they are isomorphic. Clearly an oblique morphism can be composed with a $\mathcal{C}$-morphism on the left, or with a $\mathcal{D}$-morphism on the right; it straddles the two categories, so to speak. Let us write $\mathcal{O}(X, \underline{Y})$ for the oblique morphisms from X to $\underline{Y}$. Now an adjunction from $\mathcal{C}$ to $\mathcal{D}$ can be specified by a functor $\mathcal{O} : \mathcal{C}^{\mathsf{op}} \times \mathcal{D} \to \mathbf{Set}$ (often called a *profunctor* from $\mathcal{C}$ to $\mathcal{D}$) and isomorphisms

$$\mathcal{C}(X, U\underline{Y}) \cong \mathcal{O}(X, \underline{Y}) \quad \text{natural in } X \tag{2}$$

$$\mathcal{O}(X, \underline{Y}) \cong \mathcal{D}(FX, \underline{Y}) \quad \text{natural in } \underline{Y} \tag{3}$$

Again the equivalence to the earlier definition follows from parametrized representability. To see the benefit of this definition, fix a set R and consider the adjunction

$$\mathbf{Set}(X, Y \to R) \cong \mathbf{Set}^{\mathsf{op}}(X \to R, Y) \tag{4}$$

We can decompose this isomorphism quite naturally by setting $\mathcal{O}(X, Y)$ to be $\mathbf{Set}(X \times Y, R)$.

This is essentially what is happening in CBPV+stacks: we have 3 judgements, denoting $\mathcal{C}$-morphisms (values), oblique morphisms (computations) and $\mathcal{D}$-morphisms (stacks). But the above account is overly simplistic, for, as we shall see, we want $\mathcal{D}$ to be *locally indexed* by $\mathcal{C}$.

2 Review of Call-By-Push-Value

There are two variants of CBPV: finitary and infinitely wide. In this paper we treat infinitely wide CBPV; the finitary case is treated by substituting "finite" for "countable" throughout. (The reverse substitution would not work, because for both variants contexts are finite, and hence the value category requires only finite products.)

CBPV has two disjoint classes of terms: values and computations. It likewise has two disjoint classes of types: a value has a value type, while a computation has a computation type. For clarity, we underline computation types. The types are given by

$$\begin{array}{lrl} \text{value types} & A ::= & U\underline{B} \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \\ \text{computation types} & \underline{B} ::= & FA \mid \prod_{i \in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

where I can be any countable set (finite, in finitary CBPV). The meaning of F and U is as follows. A computation of type FA *produces* a value of type A. A value of type $U\underline{B}$ is a *thunk* of a computation of type $\underline{B}$, i.e. the computation is frozen into a value so that it can be passed around. When later required, it can be *forced* i.e. executed.

Unlike in call-by-value, a function in CBPV is a computation, and hence a function type is a computation type. We will discuss this further in Sect. 3.

Like in call-by-value, an identifier in CBPV can be bound only to a value, so it must have value type. We accordingly define a *context* Γ to be a sequence

$$\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$$

of identifiers with associated value types. We often omit the identifiers and write just $A_0, \ldots, A_{n-1}$. We write $\Gamma \vdash^{\mathsf{v}} V : A$ to mean that V is a value of type A, and we write $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ to mean that M is a computation of type $\underline{B}$.

The terms of CBPV are given in Fig. 1; the symbol $\hat{\imath}$ represents any particular element of I. We explain some of the less familiar constructs. M `to` $\mathtt{x}.\ N$ is the sequenced computation that first executes M, and when this produces a value $\mathtt{x}$ proceeds to execute N. This was written in Moggi's syntax using `let`, but we reserve `let` for ordinary binding. The keyword `pm` stands for "pattern-match", and the symbol ' represents application in reverse order. Because we think of $\prod_{i\in I}$ as the type of functions taking each $i \in I$ to a computation of type $\underline{B}_i$, we have made its syntax similar to that of $\to$.

Following Lawvere [12], we say that a *context morphism* q from $\Gamma = A_0, \ldots, A_{m-1}$ to $\Delta = B_0, \ldots, B_{n-1}$ is a sequence of values $V_0, \ldots, V_{n-1}$ where $\Gamma \vdash^{\mathsf{v}} V_i : B_i$. As usual, such a morphism induces (by induction [6]) a substitution function q^* from values $\Delta \vdash^{\mathsf{v}} V : C$ to values $\Gamma \vdash^{\mathsf{v}} V : C$ and from computations $\Delta \vdash^{\mathsf{c}} M : \underline{B}$ to $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$. We define identity and composite context morphisms in the usual way.

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

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

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{force}\ V : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} (\hat{\imath}, V) : \sum_{i\in I} A_i} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i\in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} M_i : \underline{B} \quad \cdots_{\, i\in I}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\} : \underline{B}}$$

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

$$\frac{\cdots \quad \Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i \quad \cdots_{\, i\in I}}{\Gamma \vdash^{\mathsf{c}} \lambda\{\ldots, i.M_i, \ldots\} : \prod_{i\in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \prod_{i\in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} \hat{\imath}`M : \underline{B}_{\hat{\imath}}}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda\mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}}{\Gamma \vdash^{\mathsf{c}} V`M : \underline{B}}$$

Fig. 1. Terms of Call-By-Push-Value

3 The CK-Machine and $\vdash^{\mathsf{k}}$

There are several ways of presenting operational semantics for computations of CBPV. In [14] it is presented in big-step form, but here we use a *CK-machine*, in the style of [4]. We also generalize from closed computations to computations on a fixed context Γ. The machine is presented in Fig. 2, with types written explicitly. A configuration of the machine, with types, has the form

$$\Gamma \quad | \qquad M \qquad \underline{B} \qquad K \qquad \underline{C} \tag{5}$$

where $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ is the computation we are currently evaluating and K is the stack (perhaps better known as an *evaluation context*, another concept that appeared in [4]). Notice that Γ and $\underline{C}$ remain fixed throughout execution.

To begin with, we place the computation we wish to evaluate alongside the empty stack, and we apply transitions until we reach a *terminal configuration*. When the inside has the form $M\ \mathtt{to}\ \mathtt{x}.\ N$, we must first evaluate M, during which time we have no need of N. So we move the context $[\cdot]\ \mathtt{to}\ \mathtt{x}.\ N$ onto the stack, and proceed to evaluate M. Once we have evaluated M to the form $\mathtt{produce}\ V$, we remove that context from the stack and proceed to evaluate $N[V/\mathtt{x}]$. Similarly, to evaluate $V`M$ we leave the operand V on the stack while we evaluate M.

Classifying a function as a computation is a novelty of CBPV and, at first sight, seems counterintuitive. But the CK-machine provides an explanation. For $\lambda\mathtt{x}$ is treated as an instruction "pop $\mathtt{x}$" whilst $V`$ is treated as an instruction "push V". Thus a computation of type $A \to \underline{B}$ pops a value of type A and proceeds as a computation of type $\underline{B}$. Similarly, a computation of type $\prod_{i\in I}\underline{B}_i$ pops a tag $i \in I$ and proceeds as a computation of type $\underline{B}_i$.

In order to characterize the well-typed configurations, we require a judgement for stacks, of the form $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$. Thus a configuration (5) will be well-typed precisely when $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$.

By inspecting Fig. 2, we can see that the typing rules for this judgment should be as given in Fig. 3. This ensures that the well-typed configurations are precisely those obtainable from an initial configuration.

There are some evident operations on stacks.

(i) Given a context-morphism q from Γ to Δ and a stack $\Delta|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ we can *substitute* q in L to give a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} q^*K : \underline{C}$.

(ii) Given a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$, we can *dismantle* K onto M to give $\Gamma \vdash^{\mathsf{c}} M \bullet K : \underline{C}$, defined by induction on K. It is the computation whose evaluation leads to the configuration (5).

(iii) Given two stacks $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ and $\Gamma|\underline{C} \vdash^{\mathsf{k}} L : \underline{D}$, we can *concatenate* K and L to give $\Gamma|\underline{B} \vdash^{\mathsf{k}} K \mathbin{+\!\!+} L : \underline{D}$, defined by induction on K.

Lemma 3.1 *Substitution, dismantling and concatenation satisfy the following properties.*

$$\begin{aligned} \mathtt{nil} \mathbin{+\!\!+} K &= K & p^*\mathtt{nil} &= \mathtt{nil} \\ K \mathbin{+\!\!+} \mathtt{nil} &= K & p^*(K \mathbin{+\!\!+} L) &= (p^*K);(p^*L) \\ (K;L);L' &= K;(L;L') & p^*(M \bullet K) &= (p^*M) \bullet (p^*K) \end{aligned}$$

$$\begin{aligned} M \bullet \mathtt{nil} &= M & \mathsf{id}^*P &= P \\ M \bullet (K \mathbin{+\!\!+} K') &= (M \bullet K) \bullet K' & p^*q^*P &= (p;q)^*P \end{aligned}$$

Initial configuration for evaluation of $\Gamma \vdash^{\mathsf{c}} M : \underline{C}$

	$\Gamma \mid$	M	$\underline{C}$	$\mathtt{nil}$	$\underline{C}$

Transitions

	$\Gamma \mid$	$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M$	$\underline{B}$	K	$\underline{C}$
$\leadsto$	$\Gamma \mid$	$M[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$
	$\Gamma \mid$	$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\underline{B}$	K	$\underline{C}$
$\leadsto$	$\Gamma \mid$	M	FA	$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ N :: K$	$\underline{C}$
	$\Gamma \mid$	$\mathtt{produce}\ V$	FA	$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ N :: K$	$\underline{C}$
$\leadsto$	$\Gamma \mid$	$N[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$
	$\Gamma \mid$	$\mathtt{force}\ \mathtt{thunk}\ M$	$\underline{B}$	K	$\underline{C}$
$\leadsto$	$\Gamma \mid$	M	$\underline{B}$	K	$\underline{C}$
	$\Gamma \mid$	$\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\}$	$\underline{B}$	K	$\underline{C}$
$\leadsto$	$\Gamma \mid$	$M_{\hat{\imath}}[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$
	$\Gamma \mid$	$\mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M$	$\underline{B}$	K	$\underline{C}$
$\leadsto$	$\Gamma \mid$	$M[V/\mathtt{x}, V'/\mathtt{y}]$	$\underline{B}$	K	$\underline{C}$
	$\Gamma \mid$	$\hat{\imath}\text{`}M$	$\underline{B}_{\hat{\imath}}$	K	$\underline{C}$
$\leadsto$	$\Gamma \mid$	M	$\prod_{i \in I} \underline{B}_i$	$\hat{\imath} :: K$	$\underline{C}$
	$\Gamma \mid$	$\lambda\{\ldots, i.M_i, \ldots\}$	$\prod_{i \in I} \underline{B}_i$	$\hat{\imath} :: K$	$\underline{C}$
$\leadsto$	$\Gamma \mid$	$M_{\hat{\imath}}$	$\underline{B}_{\hat{\imath}}$	K	$\underline{C}$
	$\Gamma \mid$	$V\text{`}M$	$\underline{B}$	K	$\underline{C}$
$\leadsto$	$\Gamma \mid$	M	$A \to \underline{B}$	$V :: K$	$\underline{C}$
	$\Gamma \mid$	$\lambda \mathtt{x}.M$	$A \to \underline{B}$	$V :: K$	$\underline{C}$
$\leadsto$	$\Gamma \mid$	$M[V/\mathtt{x}]$	$\underline{B}$	K	$\underline{C}$

Fig. 2. CK-Machine For CBPV, With Types

$$\frac{}{\Gamma|\underline{C} \vdash^{\mathsf{k}} \mathtt{nil} \ : \underline{C}} \qquad \frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|FA \vdash^{\mathsf{k}} [\cdot] \ \mathtt{to} \ \mathtt{x}.\ M :: K : \underline{C}}$$

$$\frac{\Gamma|\underline{B}_{\hat{\imath}} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|\prod_{i \in I}\underline{B}_i \vdash^{\mathsf{k}} \hat{\imath} :: K : \underline{C}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|A \to \underline{B} \vdash^{\mathsf{k}} V :: K : \underline{C}}$$

Fig. 3. Typing rules for stacks

4 Basic Structure

4.1 Interpreting Values

Like Moggi, we interpret values in a cartesian category $\mathcal{C}$, called the "value category". As usual, the products in $\mathcal{C}$ are used for interpreting both $\times$ and context extension (comma). Although there is no value in the language $\mathtt{x} : A \times A' \vdash^{\mathsf{v}} V : A$, there will be after we extend the syntax in Sect. 6.

4.2 Interpreting Stacks

If values are interpreted in the cartesian category $\mathcal{C}$, stacks will be interpreted in a *locally* $\mathcal{C}$*-indexed category* $\mathcal{D}$. This can be defined in various ways:

- as a strict $\mathcal{C}$-indexed category (i.e. functor $\mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Cat}$) in which all the fibres $\mathcal{D}_X$ have the same set of objects $\mathsf{ob}\ \mathcal{D}$ and all the reindexing functors $\mathcal{D}_f$ are identity-on-objects
- as a $[\mathcal{C}^{\mathsf{op}}, \mathbf{Set}]$-enriched category
- using the following concrete description.

Definition 4.1 *A* locally $\mathcal{C}$-indexed category *consists of*

- *a set* $\mathsf{ob}\ \mathcal{D}$*, whose elements we call* $\mathcal{D}$-objects *and we underline, except where* $\mathsf{ob}\ \mathcal{D} = \mathsf{ob}\ \mathcal{C}$
- *for each object* $X \in \mathsf{ob}\ \mathcal{C}$ *and each pair of objects* $\underline{Y}, \underline{Z} \in \mathsf{ob}\ \mathcal{D}$*, a small set* $\mathcal{D}_X(\underline{Y}, \underline{Z})$*, an element of which we call a* $\mathcal{D}$-morphism *and write* $\underline{Y} \xrightarrow[X]{f} \underline{Z}$
- *for each object* $X \in \mathsf{ob}\ \mathcal{C}$ *and each object* $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$*, an* identity *morphism* $\underline{Y} \xrightarrow[X]{\mathsf{id}_{X,\underline{Y}}} \underline{Y}$
- *for each morphism* $\underline{Y} \xrightarrow[X]{f} \underline{Z}$ *and each morphism* $\underline{Z} \xrightarrow[X]{g} \underline{W}$*, a* composite *morphism* $\underline{Y} \xrightarrow[X]{f;g} \underline{W}$
- *for each* $\mathcal{D}$*-morphism* $\underline{Y} \xrightarrow[X]{f} \underline{Z}$ *and each* $\mathcal{C}$*-morphism* $X' \xrightarrow{k} X$*, a* reindexed $\mathcal{D}$*-morphism* $\underline{Y} \xrightarrow[X']{k^*f} \underline{Z}$

such that

$$\begin{array}{rclcrclcrcl}
\mathsf{id}; f &=& f & \quad & k^*\mathsf{id} &=& \mathsf{id} & \quad & \mathsf{id}^* f &=& f \\
f; \mathsf{id} &=& f & & k^*(f; g) &=& (k^*f); (k^*g) & & (l; k)^* f &=& l^*(k^* f) \\
(f; g); h &=& f; (g; h) & & & & & & & &
\end{array}$$

It is easy to see that this is natural for interpreting stacks: a computation type will denote an object of $\mathcal{D}$ and a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ will denote a $\mathcal{D}$-morphism over $[\![\Gamma]\!]$ from $[\![\underline{B}]\!]$ to $[\![\underline{C}]\!]$. Then identity morphisms in $\mathcal{D}$ interpret `nil`, composition interprets concatenation of stacks, and reindexing interprets substitution.

Before proceeding further, we develop some theory of locally indexed categories. Firstly, it is clear that they form a 2-category, and we have operations $-^{\mathsf{op}}$ and $\times$. The most important example of a locally $\mathcal{C}$-indexed category is called $\mathsf{self}\ \mathcal{C}$, and given by

$$\begin{array}{rcl}
\mathsf{ob}\ \mathsf{self}\ \mathcal{C} &=& \mathsf{ob}\ \mathcal{C} \\
\mathsf{self}\ \mathcal{C}_A(B, C) &=& \mathcal{C}(A \times B, C)
\end{array}$$

with the evident composition and reindexing. This is used in the following result, which appears in [17].

Proposition 4.2 *A strong monad on $\mathcal{C}$ corresponds to a monad on $\mathsf{self}\ \mathcal{C}$.*

As with ordinary categories, we wish to speak of the "homset functor" associated with $\mathcal{D}$, and we do this as follows.

Definition 4.3 (i) *Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. We write $\mathsf{opGroth}\ \mathcal{D}$ for the ordinary category where*

- *an object is a pair ${}_X\underline{Y}$ where $X \in \mathsf{ob}\ \mathcal{C}$ and $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$;*
- *a morphism from ${}_X\underline{Y}$ to ${}_{X'}\underline{Z}$ in $\mathsf{opGroth}\ \mathcal{D}$ consists of a pair ${}_kh$ where $X' \xrightarrow{k} X$ in $\mathcal{C}$ and $\underline{Y} \xrightarrow[X']{h} \underline{Z}$ in $\mathcal{D}$*
- *the identity on ${}_\Gamma\underline{X}$ is given by ${}_{\mathsf{id}}\mathsf{id}$;*
- *the composite of ${}_\Gamma\underline{X} \xrightarrow{{}_kf} {}_{\Gamma'}\underline{Y} \xrightarrow{{}_lg} {}_{\Gamma''}\underline{Z}$ is ${}_{l;k}((l^*f); g)$.*

(ii) *For a locally $\mathcal{D}$-indexed category, we write $\mathsf{Hom}_{\mathcal{D}}$, or $\mathcal{D}$ for short, for the functor*

$$\begin{array}{rcl}
\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}} \times \mathcal{D}) & \longrightarrow & \mathbf{Set} \\
{}_X(\underline{Y}, \underline{Z}) & \longmapsto & \mathcal{D}_X(\underline{Y}, \underline{Z}) \\
{}_k(f, h) & \longmapsto & \lambda g.(f; (k^*g); h)
\end{array}$$

4.3 Interpreting Computations

If we are interpreting values in a cartesian category $\mathcal{C}$, and stacks in a locally $\mathcal{C}$-indexed category $\mathcal{D}$, then we will interpret computations in a functor $\mathcal{O}$: $\mathsf{opGroth}\ \mathcal{D} \longrightarrow \mathbf{Set}$. This can be described in concrete terms.

Proposition 4.4 *A functor $\mathcal{O} : \mathsf{opGroth}\ \mathcal{D} \longrightarrow \mathbf{Set}$ is given by*

- *for each* $X \in \mathsf{ob}\ \mathcal{C}$ *and* $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$*, a small set* $\mathcal{O}_X\underline{Y}$*, an element of which we call an* $\mathcal{O}$*-morphism over* X *to* $\underline{Y}$ *and write* $g\ X \quad \underline{Y}$
- *for each* $X' \xrightarrow{k} X$ *and* $g\ X \quad \underline{Y}$ *a reindexed* $\mathcal{O}$ *morphism* $k^*g\ X' \quad \underline{Y}$
- *for each* $g\ X \quad \underline{Y}$ *and* $\underline{Y} \xrightarrow[X]{h} \underline{Y}'$ *a composite* $\mathcal{O}$ *morphism* $g;h\ X \quad \underline{Y}'$

satisfying identity, associativity and reindexing laws:

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

where g *is an oblique morphism.*

Our intention is that a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ will denote an $\mathcal{O}$-morphism over $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$ and that q^*M will denote $[\![q]\!]^*[\![M]\!]$ while the dismantling $M \bullet K$ will denote $[\![M]\!];[\![K]\!]$.

4.4 Examples

We summarize the above discussion as follows.

Definition 4.5 *A* CBPV pre-structure *consists of*

- *a cartesian category* $\mathcal{C}$
- *a locally* $\mathcal{C}$*-indexed category* $\mathcal{D}$
- *a functor* $\mathcal{O} : \mathsf{opGroth}\ \mathcal{D} \longrightarrow \mathbf{Set}$.

Here are some examples of CBPV pre-structures.

trivial Given a cartesian category $\mathcal{C}$, set $\mathcal{D}$ to be $\mathsf{self}\ \mathcal{C}$ and set $\mathcal{O}_X Y$ to be $\mathcal{C}(X, Y)$.

Scott Let $\mathcal{C}$ be **Cpo**, let a $\mathcal{D}$-object be a pointed cpo and a $\mathcal{D}$-morphism over X from $\underline{Y}$ to $\underline{Z}$ be a right-strict continuous function from $X \times \underline{Y}$ to $\underline{Z}$, and let $\mathcal{O}_X\underline{Y}$ be continuous functions from X to $\underline{Y}$.

monad Given a strong monad T on a cartesian category $\mathcal{C}$, we obtain a pre-structure $(\mathcal{C}, \mathcal{C}^T, \mathcal{O}^T)$, where a $\mathcal{C}^T$-object is a T-algebra, a $\mathcal{C}^T$-morphism over X from (Y, θ) to (Z, ϕ) is a $\mathcal{C}$-morphism $X \times Y \xrightarrow{f} Z$ satisfying

$$\begin{tikzcd} X \times TY \arrow[r, "t(X,Y)"] \arrow[d, "X \times \theta"'] & T(X \times Y) \arrow[r, "Tf"] & TZ \arrow[d, "\phi"] \\ X \times Y \arrow[rr, "f"'] & & Z \end{tikzcd}$$

and let $\mathcal{O}^T_X(Y, \theta)$ be $\mathcal{C}(X, Y)$.

storage Given a set S, let $\mathcal{C}$ be **Set**, let $\mathcal{D}$ be self **Set** and let $\mathcal{O}_X Y$ be $\mathbf{Set}(S \times X, Y)$.

erratic choice Let $\mathcal{C}$ be **Set**, let $\mathcal{D}$ be **Rel** i.e. an object is a set and a morphism over X from Y to Z is a relation from $X \times Y$ to Z, and let an $\mathcal{O}$-morphism over X to Y be a relation from X to Y.

continuations Given a set R, let $\mathcal{C}$ be $\mathbf{Set}$, let $\mathcal{D}$ be $(\mathsf{self}\ \mathbf{Set})^{\mathsf{op}}$ and let $\mathcal{O}_X Y$ be $\mathbf{Set}(X \times Y, R)$.

5 CBPV Adjunction Models

5.1 Universal Properties

So far, the only type constructors we can interpret are 1 and $\times$. The rest are interpreted using the following universal properties. The notation f^*g means $f; g$ in the case that g is a value morphism.

Definition 5.1 *In a CBPV pre-structure* $(\mathcal{C}, \mathcal{D}, \mathcal{O})$,

***U*$\underline{\boldsymbol{B}}$** *a* right adjunctive *for a* $\mathcal{D}$*-object* $\underline{B}$ *is a* $\mathcal{C}$*-object* V *and an* $\mathcal{O}$*-morphism* $\mathsf{force}_{\curlywedge}V \quad \underline{B}$, *such that the functions*

$$\begin{aligned} \mathcal{C}(X, V) &\longrightarrow \mathcal{O}_X \underline{B} && \textit{for all } X && (6) \\ f &\longmapsto f^*\mathsf{force} \end{aligned}$$

are isomorphisms

F A *a* left adjunctive *for a* $\mathcal{C}$*-object* A *is a* $\mathcal{D}$*-object* $\underline{V}$ *and an* $\mathcal{O}$*-morphism* $\mathsf{produce}_{\curlywedge}A \quad \underline{V}$, *such that the functions*

$$\begin{aligned} \mathcal{D}_X(\underline{V}, \underline{Y}) &\longrightarrow \mathcal{O}_{X \times A} \underline{Y} && \textit{for all } X, \underline{Y} && (7) \\ h &\longmapsto (\pi'^*_{X,A}\mathsf{produce}); (\pi^*_{X,A} h) \end{aligned}$$

are isomorphisms

$\sum_{i \in I} \boldsymbol{A_i}$ *a* coproduct *for a family* $\{A_i\}_{i \in I}$ *of* $\mathcal{C}$*-objects is a* $\mathcal{C}$*-object* V *and, for each* $i \in I$, *a* $\mathcal{C}$*-morphism* $A_i \xrightarrow{\mathsf{in}_i} V$, *such that the functions*

$$\begin{aligned} \mathcal{C}(X \times V, Y) &\longrightarrow \textstyle\prod_{i \in I} \mathcal{C}(X \times A_i, Y) && \textit{for all } X, Y && (8) \\ \mathcal{O}_{X \times V} \underline{Y} &\longrightarrow \textstyle\prod_{i \in I} \mathcal{O}_{X \times A_i} \underline{Y} && \textit{for all } X, \underline{Y} && (9) \\ \mathcal{D}_{X \times V}(\underline{Y}, \underline{Z}) &\longrightarrow \textstyle\prod_{i \in I} \mathcal{D}_{X \times A_i}(\underline{Y}, \underline{Z}) && \textit{for all } X, \underline{Y}, \underline{Z} && (10) \\ f &\longmapsto \lambda i.((X \times \mathsf{in}_i)^* f) \end{aligned}$$

are isomorphisms

$\prod_{i \in I} \underline{\boldsymbol{B}}_{\boldsymbol{i}}$ *a* product *for a family* $\{\underline{B}_i\}_{i \in I}$ *of* $\mathcal{D}$*-objects is a* $\mathcal{D}$*-object* $\underline{V}$ *and, for each* $i \in I$, *a* $\mathcal{D}$*-morphism* $\underline{V} \xrightarrow[1]{\pi_i} \underline{B}_i$, *such that the functions*

$$\begin{aligned} \mathcal{O}_X \underline{V} &\longrightarrow \textstyle\prod_{i \in I} \mathcal{O}_X \underline{B}_i && \textit{for all } X && (11) \\ \mathcal{D}_X(\underline{Y}, \underline{V}) &\longrightarrow \textstyle\prod_{i \in I} \mathcal{D}_X(\underline{Y}, \underline{B}_i) && \textit{for all } X, \underline{Y} && (12) \\ h &\longmapsto \lambda i.(h; ()^* \pi_i) \end{aligned}$$

are isomorphisms

$\boldsymbol{A} \to \underline{\boldsymbol{B}}$ *an* exponential *from a* $\mathcal{C}$*-object* A *to a* $\mathcal{D}$*-object* $\underline{B}$ *is a* $\mathcal{D}$*-object* $\underline{V}$ *and a* $\mathcal{D}$*-morphism* $\underline{V} \xrightarrow[A]{\mathsf{ev}} \underline{B}$, *such that the functions*

$$\mathcal{O}_X\underline{V} \longrightarrow \mathcal{O}_{X\times A}\underline{B} \qquad \textit{for all } X \tag{13}$$
$$\mathcal{D}_X(\underline{Y},\underline{V}) \longrightarrow \mathcal{D}_{X\times A}(\underline{Y},\underline{B}) \qquad \textit{for all } X,\underline{Y} \tag{14}$$
$$h \longmapsto (\pi^*_{X,A}h);(\pi'^*_{X,A}\mathsf{ev})$$

are isomorphisms.

Definition 5.2 *A* strong adjunction *is a CBPV pre-structure* $(\mathcal{C},\mathcal{D},\mathcal{O})$ *with all right adjunctives and left adjunctives—we say that it goes from* $\mathcal{C}$ *to* $\mathcal{D}$.

As a variation of the characterization of adjunctions in Sect. 1.2, we have

Proposition 5.3 *[15] A strong adjunction from* $\mathcal{C}$ *to* $\mathcal{D}$ *is equivalent to an adjunction from* self $\mathcal{C}$ *to* $\mathcal{D}$.

Thus a strong adjunction from $\mathcal{C}$ gives rise to a monad on self $\mathcal{C}$ i.e. a strong monad on $\mathcal{C}$; hence the word "strong".

Definition 5.4 *A* CBPV adjunction model *is a strong adjunction* $(\mathcal{C},\mathcal{D},\mathcal{O})$ *with all countable coproducts, countable products and exponentials. We write* $U,F,\to$ *etc. for the operations on objects.*

It is obvious how to define the interpretation of CBPV in such a structure. We mention that part of Def. 5.4 is redundant, notably the requirement for (9),(11),(13) to be isomorphisms, but the requirement for (10) to be an isomorphism does not appear to be redundant. **Warning** The definition of CBPV adjunction model in [15] mistakenly omitted this requirement.

Proposition 5.5 *Every CBPV adjunction model* $(\mathcal{C},\mathcal{D},\mathcal{O},\ldots)$ *is isomorphic to one in which the* thunk/force *isomorphism (6) is the identity.*

Proof. Leave the value category and stack category unchanged, and set $\mathcal{O}'$ to be the functor taking ${}_X\underline{Y}$ to $\mathcal{C}(X,U\underline{Y})$.

Despite this result, there are many semantics, such as storage, continuation and game models, which it is more computationally natural to present in a form where (6) is not the identity.

5.2 Examples

We first require the following, adapted from [1,2].

Definition 5.6 *A* distributive coproduct *for a family of objects* $\{A_i\}_{i\in I}$ *in a cartesian category* $\mathcal{C}$ *is an object* V *and a* $\mathcal{C}$*-morphism* $A_i \xrightarrow{\mathsf{in}_i} V$ *for each* $i \in I$*, such that the functions*

$$\mathcal{C}(X\times V,Y) \longrightarrow \textstyle\prod_{i\in I}\mathcal{C}(X\times A_i,Y) \qquad \textit{for all } X,Y \tag{15}$$
$$f \longmapsto \lambda i.((X\times \mathsf{in}_i);f)$$

are isomorphisms. (More abstractly, it is a coproduct in self $\mathcal{C}$*.) A cartesian category with all countable distributive coproducts is called a* countably distributive category.

Any distributive coproduct is a coproduct in $\mathcal{C}$ (conversely if $\mathcal{C}$ is a ccc).

We now look again at the examples of CBPV pre-structures described in Sect. 4.4 to see when they have all the required universal elements.

trivial The pre-structure given by a cartesian category $\mathcal{C}$ is a CBPV model iff $\mathcal{C}$ is *countably bicartesian closed*, i.e. a ccc with all countable coproducts and products. Such models are called *trivial* and they interpret CBPV with no computational effects at all. Indeed, "CBPV adjunction model" can be seen as a generalization of "countably bicartesian closed category" to accommodate computational effects.

Scott This is clearly a CBPV adjunction model.

monad The pre-structure given by a strong monad T on a cartesian category $\mathcal{C}$ is a CBPV model iff

- $\mathcal{C}$ is countably distributive
- $\mathcal{C}$ has a product for every countable family of T-algebra carriers
- $\mathcal{C}$ has an exponential from every object to every T-algebra carrier.

Notice that this implies that $\mathcal{C}$ has all Kleisli exponentials and all countable products of Kleisli exponentials.

storage This pre-structure is clearly a CBPV adjunction model, see below. More generally, given a CBPV model $(\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$ and a $\mathcal{C}$-object S, we obtain another CBPV model by setting $\mathcal{C}'$ to be $\mathcal{C}$, setting $\mathcal{D}'$ to be $\mathcal{D}$, and setting $\mathcal{O}'_X \underline{Y}$ to be $\mathcal{O}_{S \times X} \underline{Y}$.

erratic choice This pre-structure is clearly a CBPV adjunction model, see below.

continuations This pre-structure is clearly a CBPV adjunction model, see below. More generally, given a CBPV model $(\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$ and a $\mathcal{D}$-object $\underline{R}$, we obtain another CBPV model by setting $\mathcal{C}'$ to be $\mathcal{C}$, setting $\mathcal{D}'$ to be $(self\mathcal{C})^{\mathsf{op}}$, and setting $\mathcal{O}'_X Y$ to be $\mathcal{O}_{X \times Y} \underline{R}$.

The connectives in these examples are given in the following table:

model	$\sum_{i \in I}$	1	$\times$	U	F	$\prod_{i \in I}$	$\rightarrow$
Scott	$\sum_{i \in I}$	1	$\times$	$-$	$-_\perp$	$\prod_{i \in I}$	$\rightarrow$
storage	$\sum_{i \in I}$	1	$\times$	$S \rightarrow -$	$S \times -$	$\prod_{i \in I}$	$\rightarrow$
storage, general	$\sum_{i \in I}$	1	$\times$	$U(S \rightarrow -)$	$F(S \times -)$	$\prod_{i \in I}$	$\rightarrow$
erratic choice	$\sum_{i \in I}$	1	$\times$	$\mathcal{P}$	$-$	$\sum_{i \in I}$	$\times$
continuations	$\sum_{i \in I}$	1	$\times$	$- \rightarrow R$	$- \rightarrow R$	$\sum_{i \in I}$	$\times$
continuations, general	$\sum_{i \in I}$	1	$\times$	$U(- \rightarrow \underline{R})$	$U(- \rightarrow \underline{R})$	$\sum_{i \in I}$	$\times$

Notice the we recover Moggi's strong monad in each case.

6 The Equational Theory

6.1 Complex Values and Stacks

Since we want the term model of CBPV+stacks to be an adjunction model, we require additional syntax—otherwise we cannot even form a projection $\mathtt{x} : A \times A' \vdash^{\mathsf{v}} V : A$ to make a cartesian category of values. The rules for the formation of "complex" values and stacks using binding and pattern-matching are given in Fig. 4. We shall see in Sect. 6.2 that these new constructs can always be eliminated from a computation. For example, $\mathtt{produce}\ (\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W$ can be simplified into $\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\mathtt{produce}\ W$.

We explain the `where` construct as follows. In any stack K to $\underline{B}$ there is a unique occurrence of `nil`, and we can "bind" it to a stack L from $\underline{B}$, giving the concatenated stack $K{+\!\!+}L$. A stack L' from $A \to \underline{B}$ is typically of the form $V :: L$, where V is a value of type A and L is a stack from $\underline{B}$; so it can be pattern-matched as $\mathtt{x} : \mathtt{nil}$ in K.

We mention that omitting the `let` construct and the K `where nil is` L construct would not affect the theory/model equivalence of Sect. 7.2; they are only a convenience. The pattern-match constructs, on the other hand, are essential.

6.2 Properties of Equational Theory

The equational theory is the least congruence on terms-in-context containing the laws in Fig. 5. To reduce clutter, we omit the assumptions necessary to make each equation well-typed, and we employ the *bound/unbound* convention: when, in an equation—such as the η-law $M = \lambda\mathtt{x}.(\mathtt{x}`M)$—the term $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ occurs both in the scope of an $\mathtt{x}$-binder and not in the scope of an $\mathtt{x}$-binder, we assume $\mathtt{x} \notin \Gamma$. We do not write the weakening explicitly.

Lemma 6.1 *Provable equality is closed under substitution, dismantling and concatenation.*

Proof. Straightforward induction in each case. □

Proposition 6.2 *For any computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$, we can obtain a computation $\Gamma \vdash^{\mathsf{c}} M' : \underline{B}$ that does not use the complex value constructs of Sect. 6, and a proof that $M = M'$ in the equational theory. Similarly, we can eliminate these constructs from any* closed *value $\vdash^{\mathsf{v}} V : A$.*

Prop. 6.2, which does not involve stacks, is proved in [15].

One noteworthy equation of Fig. 5 is the η-law

$$M \bullet K = M\ \mathtt{to}\ \mathtt{x}.\ ((\mathtt{produce}\ \mathtt{x}) \bullet K) \tag{16}$$

This is equivalent to the two equations

$$M = M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{produce}\ \mathtt{x} \tag{17}$$

$$(M\ \mathtt{to}\ \mathtt{x}.\ N) \bullet K = M\ \mathtt{to}\ \mathtt{x}.\ (N \bullet K) \tag{18}$$

Complex Values

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

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{v}} W_i : B \quad \cdots_{\,i \in I}}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).W_i, \ldots\} : B}$$

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

Complex Stacks

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x}.A | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | \underline{B} \vdash^{\mathsf{k}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ K : \underline{C}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i | \underline{B} \vdash^{\mathsf{k}} K_i : \underline{C} \quad \cdots_{\,i \in I}}{\Gamma | \underline{B} \vdash^{\mathsf{k}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).K_i, \ldots\} : \underline{C}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | \underline{B} \vdash^{\mathsf{k}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).K : \underline{C}}$$

$$\frac{\Gamma | \underline{C} \vdash^{\mathsf{k}} K : \underline{B} \quad \Gamma | \underline{B} \vdash^{\mathsf{k}} L : \underline{D}}{\Gamma | \underline{C} \vdash^{\mathsf{k}} K\ \mathtt{where}\ \mathtt{nil}\ \mathtt{is}\ L : \underline{D}}$$

$$\frac{\cdots \quad \Gamma | \underline{C} \vdash^{\mathsf{k}} K_i : \underline{B}_i \quad \cdots_{\,i \in I} \quad \Gamma | \prod_{i \in I} \underline{B}_i \vdash^{\mathsf{k}} L : \underline{D}}{\Gamma | \underline{C} \vdash^{\mathsf{k}} \{\ldots, K_i\ \mathtt{where}\ i :: \mathtt{nil}, \ldots\}\ \mathtt{is}\ L : \underline{D}}$$

$$\frac{\Gamma, \mathtt{x} : A | \underline{C} \vdash^{\mathsf{k}} K : \underline{B} \quad \Gamma | A \to \underline{B} \vdash^{\mathsf{k}} L : \underline{D}}{\Gamma | \underline{C} \vdash^{\mathsf{k}} K\ \mathtt{where}\ \mathtt{x} :: \mathtt{nil}\ \mathtt{is}\ L : \underline{D}}$$

Fig. 4. Complex Values and Stacks

(18) makes it clear why a stack denotes an algebra homomorphism in the monad models of CBPV. As instances of (18) we have many familiar CBPV equations:

$$\begin{aligned}(M\ \mathtt{to}\ \mathtt{x}.\ N)\ \mathtt{to}\ \mathtt{y}.\ P &= M\ \mathtt{to}\ \mathtt{x}.\ (N\ \mathtt{to}\ \mathtt{y}.\ P)\\ \lambda\{\ldots, i.(M\ \mathtt{to}\ \mathtt{x}.\ N_i), \ldots\} &= M\ \mathtt{to}\ \mathtt{x}.\ \lambda\{\ldots, i.N_i, \ldots\}\\ \lambda \mathtt{y}.(M\ \mathtt{to}\ \mathtt{x}.\ N) &= M\ \mathtt{to}\ \mathtt{x}.\ \lambda \mathtt{y}.N\end{aligned}$$

In [15] rather complex lemmas were required to prove these valid in the categorical semantics. It is an advantage of working with stacks that these equations become straightforward.

β-laws

$$
\begin{array}{rcl}
\texttt{let}\ V\ \texttt{be x.}\ Q & = & Q[V/\texttt{x}] \\
K\ \texttt{where nil is}\ L & = & K +\!\!+ L \\
\texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \{\ldots, (i, \texttt{x}).Q_i, \ldots\} & = & Q_{\hat{\imath}}[V/\texttt{x}] \\
\texttt{pm}\ (V, V')\ \texttt{as}\ (\texttt{x}, \texttt{y}).Q & = & Q[V/\texttt{x}, V'/\texttt{y}] \\
\texttt{force thunk}\ M & = & M \\
(\texttt{produce}\ V)\ \texttt{to x.}\ M & = & M[V/\texttt{x}] \\
\hat{\imath}`\lambda\{\ldots, i.M_i, \ldots\} & = & M_{\hat{\imath}} \\
\{\ldots, K_i\ \texttt{where}\ i :: \texttt{nil}, \ldots\}\ \texttt{is}\ \hat{\imath} :: L & = & K_{\hat{\imath}} +\!\!+ L \\
V`\lambda\texttt{x}.M & = & M[V/\texttt{x}] \\
K\ \texttt{where x :: nil is}\ V :: L & = & K[V/\texttt{x}] +\!\!+ L
\end{array}
$$

η-laws

$$
\begin{array}{rcl}
Q[V/\texttt{z}] & = & \texttt{pm}\ V\ \texttt{as}\ \{\ldots, (i, \texttt{x}).Q[(i, \texttt{x})/\texttt{z}], \ldots\} \\
Q[V/\texttt{z}] & = & \texttt{pm}\ V\ \texttt{as}\ (\texttt{x}, \texttt{y}).Q[(\texttt{x}, \texttt{y})/\texttt{z}] \\
V & = & \texttt{thunk force}\ V \\
M \bullet K & = & M\ \texttt{to x.}\ ((\texttt{produce x}) \bullet K) \\
K +\!\!+ L & = & [\cdot]\ \texttt{to x.}\ ((\texttt{produce x}) \bullet K) :: L \\
M & = & \lambda\{\ldots, i.i`M, \ldots\} \\
K +\!\!+ L & = & \{\ldots, (K +\!\!+ i :: \texttt{nil})\ \texttt{where}\ i :: \texttt{nil}, \ldots\}\ \texttt{is}\ L \\
M & = & \lambda\texttt{x}.(\texttt{x}`M) \\
K +\!\!+ L & = & (K +\!\!+ \texttt{x} :: \texttt{nil})\ \texttt{where x :: nil is}\ L
\end{array}
$$

Fig. 5. Equational laws for CBPV + stacks

7 General Theories

7.1 Signatures

We define a *sequent* to be a judgment without a term; thus a sequent is of the form $\Gamma \vdash^{\mathsf{v}} B$ or $\Gamma \vdash^{\mathsf{c}} \underline{B}$ or $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{B}'$. Here Γ is a finite sequence $A_0, \ldots, A_{r-1}$ of value types, with no associated identifiers.

In order to follow the approach of [11], we need the facility to add primitive operations to CBPV. Each operation has a "sorting" which is a sequent [1]. For example, the sorting + is $\texttt{nat}, \texttt{nat} \vdash^{\mathsf{v}} \texttt{nat}$. Applied to two values of type nat, it makes a value of type nat. An operation div that produces a natural number, or—if the divisor is zero—raises an error, has the sorting $\texttt{nat}, \texttt{nat} \vdash^{\mathsf{c}} F\texttt{nat}$.

More distinctively, we could have stack-like operations that build a computation M into a computation $f(V_0, \ldots, V_{r-1}|M)$ whose execution begins by

[1] In [11], the only primitive operations required are constants, because in simply typed λ-calculus every sequent is equivalent to a closed sequent, but in CBPV that is not true for value sequents.

executing M, and so the context $f(V_0,\ldots,V_{r-1}|[\cdot])$ is placed on to the stack for future use.

$$\begin{array}{lllrl} & \Gamma \mid f(V_0,\ldots,V_{r-1}|M) & \underline{B}' & K & \underline{C} \\ \leadsto & \Gamma \mid M & \underline{B} & f(V_0,\ldots,V_{r-1}|[\cdot]) :: K & \underline{C} \end{array}$$

A *signature* is a collection of primitive operations; more formally, a function from sequents to sets. The associated rules are shown in Fig. 6. We have, as an instance of (18), the equation

$$f(V_0,\ldots,V_{r-1}|M \ \texttt{to}\ \texttt{x}.\ N) = M \ \texttt{to}\ \texttt{x}.\ f(V_0,\ldots,V_{r-1}|N)$$

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

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

$$\frac{\Gamma \vdash^{\mathsf{v}} V_0 : A_0 \quad \cdots \quad \Gamma \vdash^{\mathsf{v}} V_{r-1} : A_{r-1} \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} f(V_0,\ldots,V_{r-1}|M) : \underline{B}'}\ f \in S\,(A_0,\ldots,A_{r-1}|\underline{B} \vdash^{\mathsf{k}} \underline{B}')$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V_0 : A_0 \quad \cdots \quad \Gamma \vdash^{\mathsf{v}} V_{r-1} : A_{r-1} \quad \Gamma|\underline{B}' \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|\underline{B} \vdash^{\mathsf{k}} f(V_0,\ldots,V_{r-1}|[\cdot]) :: K : \underline{C}}\ f \in S\,(A_0,\ldots,A_{r-1}|\underline{B} \vdash^{\mathsf{k}} \underline{B}')$$

Fig. 6. Rules For A Signature S

7.2 Theory/Model Equivalence

We are now in a position to state a theory/model equivalence theorem. Our formulation of this theorem (though not the proof) follows [11], in particular the use of structure preservation on the nose. For the purposes of this section, we insist on defining a cartesian category to be a category with distinguished terminal object and binary products. We cannot define it to be a category with distinguished n-ary products for all $n \in \mathbb{N}$, because the operations on objects and on types must be *identical* for the on-the-nose approach to work. This is a flaw, pervasive in categorical semantics, and we leave its rectification to future work.

We begin our account by defining an *on-the-nose morphism* from CBPV adjunction model $(\mathcal{C},\mathcal{D},\mathcal{O},\ldots)$ to CBPV adjunction model $(\mathcal{C}',\mathcal{D}',\mathcal{O}',\ldots)$ in the obvious way. We then define **Adj** to be the category of CBPV adjunction models and on-the-nose morphisms.

A *theory* for CBPV+stacks consists of

- a type structure τ, i.e. two (not necessarily small) sets $\mathsf{valtypes}\ \tau$ and $\mathsf{comptypes}\ \tau$, equipped with a binary operation $\times$ on $\mathsf{valtypes}\ \tau$, and similarly with operations for all the other connectives—note that τ is not required to be freely generated
- a τ-signature S, i.e. a function from sequents in τ to sets
- a congruence $\equiv$ on the terms-in-context in τ generated from S according to Fig. 1, 4 and 6, containing all the equations of Fig. 5, and closed under substitution, dismantling and concatenation.

Remark 7.1 An alternative formulation is followed in [8,13], avoiding the mention of congruence. For fixed type structure τ, it is clear that the terms-in-context on a given τ-signature S, quotiented by the equations of Fig. 5, form another τ-signature; we write this as $T_\tau S$. Moreover, we can extend T_τ to a monad on the category $\mathcal{S}_\tau$ of τ-signatures. (This makes use of Lemma. 6.1.) We then define a *direct model* to be a type structure τ together with an algebra for the monad T_τ. This is equivalent to a theory.

An *on-the-nose morphism* F from a theory $(\tau, S, \equiv)$ to a theory $(\tau', S', \equiv')$ provides

- a function from value types of τ to value types of τ', and similarly for computation types, preserving all connectives
- a function from $\equiv$-equivalence classes of S-values $A_0, \ldots, A_{m-1} \vdash^{\mathsf{v}} V : B$ to $\equiv'$-equivalence classes of S'-values $FA_0, \ldots, FA_{m-1} \vdash^{\mathsf{v}} W : FB$, and similarly for computations and stacks, preserving all the term constructors.

We can prove by induction that F must preserve substitution, dismantling and concatenation. We write **Th** for the category of theories and on-the-nose morphisms, and can now formulate our main result.

Proposition 7.2 *The categories* **Adj** *and* **Th** *are equivalent.*

Proof. Let $\mathcal{A} = (\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$ be an adjunction model. We define its *internal language* $\mathbf{L}\mathcal{A}$ to be the following theory. The type structure τ is given by $\mathsf{ob}\ \mathcal{C}$ and $\mathsf{ob}\ \mathcal{D}$. The signature S is given by

$$
\begin{aligned}
S(A_0, \ldots, A_{r-1} \vdash^{\mathsf{v}} B) &= \mathcal{C}(A_0 \times \cdots \times A_{r-1}, B) \\
S(A_0, \ldots, A_{r-1} \vdash^{\mathsf{c}} \underline{B}) &= \mathcal{O}_{A_0 \times \cdots \times A_{r-1}} \underline{B} \\
S(A_0, \ldots, A_{r-1} | \underline{B} \vdash^{\mathsf{k}} \underline{C}) &= \mathcal{C}_{A_0 \times \cdots \times A_{r-1}}(\underline{B}, \underline{C})
\end{aligned}
$$

We state rather pedantically that we define n-ary product in $\mathcal{C}$ by left association, so the product of the singleton sequence A is $1 \times A$.

We next interpret the terms generated by S in $\mathcal{A}$ in the obvious way. For example, given $\Gamma \vdash^{\mathsf{v}} M : FA$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}$, we interpret M $\mathtt{to}$ $\mathtt{x}.\ N$ as $M; r^A_{[\![\Gamma]\!],[\![\underline{B}]\!]} N$, where r^A is the isomorphism for the left adjunctive of A.

We prove that $M \bullet K$ denotes $[\![M]\!]; [\![K]\!]$ and similarly for substitution and concatenation. These are all straightforward inductions (the proof for substitution requires us to prove that all the isomorphisms in Def. 5.1 are natural in X). We therefore see that the kernel of the semantics preserves substitution, dismantling and concatenation; we set $\equiv$ to be this congruence. Showing that it satisfies all the required equational laws is straightforward.

In the opposite direction, given a theory $L = (\tau, S, \equiv)$ we define its *classifying model* $\mathbf{C}L$ to be the following adjunction model. The objects are just the types of τ, and the operations on objects given by the type structure. The homsets are defined by

$$\begin{aligned}
\mathcal{C}(A, B) &= S(A \vdash^{\mathsf{v}} B) \\
\mathcal{O}_A \underline{B} &= S(A \vdash^{\mathsf{c}} \underline{B}) \\
\mathcal{C}_A(\underline{B}, \underline{C}) &= S(A|\underline{B} \vdash^{\mathsf{k}} \underline{C})
\end{aligned}$$

All the categorical operations are defined in the obvious way, and proving the equations of an adjunction model is trivial, using Lemma 3.1 generalized to terms generated by S.

Next we have to show these two operations $\mathbf{L}$ and $\mathbf{C}$ to be inverse up to on-the-nose isomorphism. Given a model $\mathcal{A} = (\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$, we want to construct an on-the-nose isomorphism $\alpha_{\mathcal{A}}$ from $\mathcal{A}$ to $\mathbf{CL}\mathcal{A} = (\mathcal{C}', \mathcal{D}', \mathcal{O}')$. These two models have the same objects and operations on objects, and we set $\alpha_{\mathcal{A}}$ to be identity on objects. The homsets of $\mathcal{A}'$ are given by

$$\begin{aligned}
\mathcal{C}'(A, B) &= \mathcal{C}(1 \times A, B) \\
\mathcal{O}'_A \underline{B} &= \mathcal{O}_{1 \times A} \underline{B} \\
\mathcal{D}'_A(\underline{B}, \underline{C}) &= \mathcal{D}'_{1 \times A}(\underline{B}, \underline{C})
\end{aligned}$$

and we set $\alpha_{\mathcal{A}}$ to reindex by the isomorphism from $1 \times A$ to A. Proving that this is structure-preserving is long and straightforward.

On the other hand, given a theory $\mathcal{L} = (\tau, S, \equiv)$, we want to construct an on-the-nose isomorphism $\beta_{\mathcal{L}}$ from $\mathcal{L}$ to $\mathbf{LC}\mathcal{L} = (\tau, S', \equiv')$. The two theories have the same type structure, and we set $\beta_{\mathcal{L}}$ to be identity on objects. The signature S' is given by

$$\begin{aligned}
S'(A_0, \ldots, A_{r-1} \vdash^{\mathsf{v}} B) &= S(A_0 \times \cdots \times A_{r-1} \vdash^{\mathsf{v}} B) \\
S'(A_0, \ldots, A_{r-1} \vdash^{\mathsf{c}} \underline{B}) &= S(A_0 \times \cdots \times A_{r-1} \vdash^{\mathsf{c}} \underline{B}) \\
S'(A_0, \ldots, A_{r-1}|\underline{B} \vdash^{\mathsf{k}} \underline{C}) &= S(A_0 \times \cdots \times A_{r-1}|\underline{B} \vdash^{\mathsf{k}} \underline{C})
\end{aligned}$$

and we set $\beta_{\mathcal{L}}$ to reindex by the obvious context morphism from $A_0 \times \cdots \times A_{r-1}$ to $A_0, \ldots, A_{r-1}$. Proving that this is structure preserving is straightforward.

It is obvious how to extend $\mathbf{L}$ and $\mathbf{C}$ to functors between $\mathbf{Adj}$ and $\mathbf{Th}$, and it is then easily seen that α and β are natural. $\square$

8 Call-By-Value is Kleisli, Call-By-Name is co-Kleisli

Given a CBPV adjunction model, we can form Kleisli and co-Kleisli adjunctions and obtain fully faithful comparison functors.

$$
\begin{tikzcd}
& \text{self } \mathcal{C} \arrow[rr, shift left, "F"] \arrow[rr, phantom, "\perp"] \arrow[d, shift left] \arrow[d, phantom, "\vdash"'] & & \mathcal{D} \arrow[ll, shift left, "U"] \arrow[d, shift left] \arrow[d, phantom, "\dashv"] & \\
\text{Kleisli} & \mathcal{C}_{UF} \arrow[u, shift left] \arrow[urr, dashed] & & \mathcal{D}_{FU} \arrow[u, shift left] \arrow[ull, dashed] & \text{co-Kleisli}
\end{tikzcd}
$$

For the Scott model, the Kleisli category (over 1) provides the partial maps model for CBV [18], while the co-Kleisli category (over 1) is pointed cpos and continuous maps—the standard model for CBN. In general:

- A CBV term is translated into a CBPV computation of the form $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : FB$. Hence it is interpreted in the Kleisli category. In fact we have a strong adjunction between $\mathcal{C}$ and $\mathcal{C}_{UF}$, in the terminology of [16] this is a *strong κ-category*.
- A CBN term is translated into a CBPV computation of the form $U\underline{A}_0, \ldots, U\underline{A}_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$. Hence it is interpreted in the co-Kleisli category, in the fibre over 1.

Notice that in a continuation model, the duality between the Kleisli and co-Kleisli categories [19,20] is a consequence of the duality between $\mathcal{C}$ and $\mathcal{D}_1$.

9 Further Work

Whilst we have attached names to the various adjunction models such as "storage", "erratic choice" and so forth, these names need to be justified and explained, especially as regards the $\mathcal{D}$-morphisms. It remains also to describe more advanced models such as possible world and game models. Game models without a bracketing condition [9] are known to be continuation models [10], and hence must give adjunction models; but for well-bracketed game models [7] the situation is less immediate.

An interesting question is whether the universal properties of Sect. 5.1 can be described in terms of naturality rather than elements, by means of a Yoneda Lemma. A difficulty here is the requirement for (10) to be an isomorphism.

Macros used here include Paul Taylor's diagram macros.

References

[1] Carboni, A., S. Lack and R. F. C. Walters, *Introduction to extensive and distributive categories*, Journal of Pure and Applied Algebra **84** (1993), pp. 145–158.

[2] Cockett, J. R. B., *Introduction to distributive categories*, Mathematical Structures in Computer Science **3** (1993), pp. 277–307.

[3] Curien, P.-L. and H. Herbelin, *The duality of computation.*, in: *Proceedings of the ACM Sigplan International Conference on Functional Programming (ICFP-00)*, ACM Sigplan Notices **35.9** (2000), pp. 233–243.

[4] Felleisen, M. and D. P. Friedman, *Control operators, the SECD-machine, and the λ-calculus*, in: M. Wirsing, editor, *Formal Description of Programming Concepts*, North-Holland, 1986 pp. 193–217.

[5] Filinski, A., "Controlling Effects," Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania (1996).

[6] Fiore, M., G. D. Plotkin and D.Turi, *Abstract syntax and variable binding*, in: G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)* (1999), pp. 193–202.

[7] Hyland, J. M. E. and C.-H. L. Ong, *On full abstraction for PCF: I, II, and III*, Information and Computation **163** (2000), pp. 285–408.

[8] Jeffrey, A., *A fully abstract semantics for a higher-order functional language with nondeterministic computation*, TCS: Theoretical Computer Science **228** (1999).

[9] Laird, J., *Full abstraction for functional languages with control*, in: *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, Warsaw, Poland, 1997, pp. 58–67.

[10] Laird, J., "A Semantic Analysis of Control," Ph.D. thesis, University of Edinburgh (1998).

[11] Lambek, J. and P. Scott, "Introduction to Higher Order Categorical Logic," Cambridge University Press, Cambridge, 1986.

[12] Lawvere, F. W., "Functional Semantics of Algebraic Theories," Ph.D. thesis, Columbia University (1963).

[13] Levy, P. B., *λ-calculus and cartesian closed categories* (1996), essay for Part III of the Mathematical Tripos, Cambridge University, manuscript.

[14] Levy, P. B., *Call-by-push-value: a subsuming paradigm (extended abstract)*, in: J.-Y. Girard, editor, *Typed Lambda-Calculi and Applications*, LNCS **1581** (1999), pp. 228–242.

[15] Levy, P. B., "Call-by-push-value," Ph.D. thesis, Queen Mary, University of London (2001).

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

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

[18] Plotkin, G. D., *Lectures on predomains and partial functions* (1985), course notes, Center for the Study of Language and Information, Stanford.

[19] Selinger, P., *Control categories and duality: On the categorical semantics of the lambda-mu calculus*, Mathematical Structures in Computer Science **11** (2001), pp. 207–260.

[20] Streicher, T. and B. Reus, *Classical logic, continuation semantics and abstract machines*, Journal of Functional Programming **8** (1998), pp. 543–572.