Theory and Applications of Categories, Vol. 14, No. 5, 2005, pp. 75–110.

ADJUNCTION MODELS FOR CALL-BY-PUSH-VALUE WITH STACKS

PAUL BLAIN LEVY

ABSTRACT. Call-by-push-value is a "semantic machine code", providing a set of simple primitives from which both the call-by-value and call-by-name paradigms are built. We present its operational semantics as a stack machine, suggesting a term judgement of stacks. We then see that CBPV, incorporating these stack terms, has a simple categorical semantics based on an adjunction between values and stacks. There are no coherence requirements.

We describe this semantics incrementally. First, we introduce locally indexed categories and the opGrothendieck construction, and use these to give the basic structure for interpreting the three 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 see a wide range of instances of this structure: we give examples for divergence, storage, erratic choice, continuations, possible worlds and games (with or without a bracketing condition), in each case resolving the strong monad from the literature into a strong adjunction. And we give ways of constructing models from other models.

Finally, we see that call-by-value and call-by-name are interpreted within the Kleisli and co-Kleisli parts, respectively, of a call-by-push-value adjunction.

1. Introduction

1.1. BACKGROUND. Moggi [Mog91] introduced the use of a *strong monad* T on a cartesian category $\mathcal{C}$ to model call-by-value languages. As noted by [Fil96] 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 [Lev99, Lev], subsuming call-by-value and call-by-name. Two key type constructors 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

An extended abstract of this paper was presented at the 9th Conference on Category Theory and Computer Science, Ottawa, 2002 [Lev03].

Received by the editors 2004-01-15 and, in revised form, 2004-11-04.

Transmitted by R. Blute. Published on 2005-03-25.

2000 Mathematics Subject Classification: 18C50.

Key words and phrases: call-by-push-value, adjunction, CK-machine, monad, denotational semantics, indexed category, continuations, possible worlds, game semantics, call-by-name, call-by-value.

$S \times -$, 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 [Lev01].

But fortunately, recent work on CK-machine semantics (a form of operational semantics [FF86]) for CBPV has brought to light a new judgement: that of *stacks*. (Independently, a stack judgement with some similar rules was introduced in [CH00], in the setting of call-by-name and call-by-value with control effects.) The categorical semantics of CBPV+stacks is precisely the adjunction structure noticed in each of the concrete models. The purpose of this paper is to present this adjunction semantics.

We will describe numerous such adjunctions, including models for divergence, storage, erratic choice, continuations, possible worlds in the style of [Lev02] and pointer games in the style of [AHM98, HO00]; as well as general ways of building these adjunctions. We do not motivate them individually in this paper, because they are given merely as examples of the categorical structure.

In the case of pointer games, it is noteworthy that our categorical structure includes not only *unbracketed* models [Lai97]—which are continuation models after all [Lai98]—but also *well-bracketed* models, which are not. (See also Remark 6.8.)

As usual in categorical semantics, we show a theory/model equivalence. In order to achieve this, we need to introduce additional constructs called *complex values* and *complex stacks*. Fortunately, however, these constructs can always be removed from a computation.

Finally, we look at some related topics, including CBV and CBN.

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 correspond. 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 $\mathcal{O} : \mathcal{C}^{\mathrm{op}} \times \mathcal{D} \to \mathbf{Set}$ (also called a left $\mathcal{C}$-, right $\mathcal{D}$-bimodule) 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 three 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 $\mathcal{C}$.

1.3. Element Style vs. Naturality Style. Universal properties in category theory can usually be defined either in terms of elements or in terms of naturality of isomorphisms. Here is a well-known example. A *product* for a family of objects $\{A_i\}_{i\in I}$ in a category $\mathcal{C}$ consists of an object V—the *vertex*—together with either of the following:

- for each $i \in I$, a morphism $V \xrightarrow{\pi_i} A_i$, such that the function

$$\begin{array}{rcll} \mathcal{C}(X, V) & \longrightarrow & \prod_{i\in I} \mathcal{C}(X, A_i) & \qquad \text{for all } X \\ f & \longmapsto & \lambda i.(f; \pi_i) & \end{array} \tag{5}$$

  is an isomorphism

- an isomorphism

$$\mathcal{C}(X, V) \;\cong\; \prod_{i\in I} \mathcal{C}(X, A_i) \qquad \text{natural in } X \tag{6}$$

The equivalence of these two definitions follows from the Yoneda Lemma.

For the universal properties we will treat in this paper, we will give definitions in element-style only, and leave the naturality-style formulation to future work—see also [Lev04]. Here is an important special case, adapted from [CLW93, Coc93] and used to interpret sum types.

1.4. Definition. 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

$$\begin{array}{rcll} \mathcal{C}(X \times V, Y) & \longrightarrow & \prod_{i\in I} \mathcal{C}(X \times A_i, Y) & \qquad \text{for all } X, Y \\ f & \longmapsto & \lambda i.((X \times \mathsf{in}_i); f) & \end{array} \tag{7}$$

are isomorphisms.

Clearly, any distributive coproduct is a coproduct; the converse holds if $\mathcal{C}$ is cartesian closed but is false in general. A cartesian category with all finite (resp. countable) distributive coproducts is called a *distributive* (resp. countably distributive) category.

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 *returns* 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. We assume formally that all terms are explicitly typed, but in this paper, to reduce clutter, we omit explicit typing information,

We explain some of the less familiar constructs as follows. M `to` $\mathtt{x}.\ N$ is the sequenced computation that first executes M, and when, this returns a value V proceeds to execute N with `x` bound to V. This was written in Moggi's syntax using `let`, but we reserve `let` for mere 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 [Law63], 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 [FPD99]) 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.

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

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

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

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

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

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

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

Figure 1: Terms of Call-By-Push-Value

3. The CK-Machine and the Stack Judgement

The operational semantics of CBPV is given in [Lev99] in big-step form, but here we use a *CK-machine*, a style of semantics introduce in [FF86] for CBV—there are many similar formulations [Kri85, PS98, SR98]. We describe it for computations on a fixed context Γ (by contrast with big-step semantics, which is defined for closed computations only). The machine is presented in Fig. 2, with types written explicitly. A configuration of the machine, with types, has the form

$$\Gamma \quad | \qquad M \qquad \underline{B} \qquad K \qquad \underline{C} \tag{8}$$

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 [FF86]). 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 computation has the form M `to x.` N, we must first evaluate M, during which time we have no need of N. So we move the context $[\cdot]$ `to x.` N onto the stack, and proceed to evaluate M. Once we have evaluated M to the form `return` V, we remove that context from the stack and proceed to evaluate $N[V/\mathtt{x}]$. Similarly, to evaluate $V\text{`}M$ 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 `x`" whilst $V\text{`}$ 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 (8) 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.

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

2. 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 (8). Think of this operation as "running the CK-machine backwards".

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

$$
\begin{array}{llllr l}
 & \Gamma \mid & M & \underline{C} & \mathtt{nil} & \underline{C}
\end{array}
$$

Transitions

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

Figure 2: CK-Machine For CBPV, With Types

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

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

Figure 3: Typing rules for stacks

3. 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*[1] K and L to give $\Gamma|\underline{B} \vdash^{\mathsf{k}} K +\!\!+ L : \underline{D}$. We can regard concatenation as a form of substitution, because $K +\!\!+ L$ is $K[L/\mathtt{nil}]$.

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

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{nil} +\!\!+ K & = & K & p^*\mathtt{nil} & = & \mathtt{nil} \\
K +\!\!+ \mathtt{nil} & = & K & p^*(K +\!\!+ L) & = & (p^*K) +\!\!+ (p^*L) \\
(K +\!\!+ L) +\!\!+ L' & = & K +\!\!+ (L +\!\!+ L') & p^*(M \bullet K) & = & (p^*M) \bullet (p^*K) \\
\\
M \bullet \mathtt{nil} & = & M & \mathsf{id}^*P & = & P \\
M \bullet (K +\!\!+ K') & = & (M \bullet K) \bullet K' & (p;q)^*P & = & p^*q^*P
\end{array}$$

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 of the form $\mathtt{x} : A \times A' \vdash^{\mathsf{v}} V : A$ (for general A, A'), there will be after we extend the syntax in Sect. 7.

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.

[1] The reader who finds the idea of concatenating stacks to be strange need not worry, because concatenation and dismantling are never performed in the operational semantics. It is only for *reasoning* about stacks that we consider these operations.

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

- a set $\mathsf{ob}\,\mathcal{D}$ of *$\mathcal{D}$-objects* which 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 $\mathcal{D}_X(\underline{Y}, \underline{Z})$ of *$\mathcal{D}$-morphisms* written $\underline{Y} \xrightarrow[X]{f} \underline{Z}$
- 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

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

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, mentioned in [Mog91].

4.4. Proposition. 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 using the "opGrothendieck construction" as follows.

4.5. Definition.

1. Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. We write $\mathsf{opGr}\ \mathcal{D}$ 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 $\mathsf{opGr}\ \mathcal{D}$ 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 ${}_{l;k}((l^*f);g)$.

2. For a locally $\mathcal{D}$-indexed category, we write $\mathsf{Hom}_{\mathcal{D}}$, or $\mathcal{D}$ for short, for the functor

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

4.6. 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 $\mathcal{O} : \mathsf{opGr}\ \mathcal{D} \longrightarrow \mathbf{Set}$, also called a right $\mathcal{D}$-module. This can be described in concrete terms.

4.7. Definition. A (locally $\mathcal{C}$-indexed) *right $\mathcal{D}$-module* 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 $\xrightarrow[X]{g} \underline{Y}$
- for each $X' \xrightarrow{k} X$ and $\xrightarrow[X]{g} \underline{Y}$ a reindexed $\mathcal{O}$ morphism $\xrightarrow[X']{k^*g} \underline{Y}$
- for each $\xrightarrow[X]{g} \underline{Y}$ and $\underline{Y} \xrightarrow[X]{h} \underline{Y}'$ a composite $\mathcal{O}$ morphism $\xrightarrow[X]{g;h} \underline{Y}'$

satisfying identity, associativity and reindexing laws:

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

where g is an $\mathcal{O}$-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 $[\![M]\!];[\![K]\!]$.

We summarize the above discussion as follows.

4.8. DEFINITION. A *CBPV judgement model* consists of

- a cartesian category $\mathcal{C}$
- a locally $\mathcal{C}$-indexed category $\mathcal{D}$
- a right $\mathcal{D}$-module $\mathcal{O}$.

4.9. EXAMPLES. Here are some examples of CBPV judgement models. We do not motivate them individually, because they are given only as examples of the structure.

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

cpo Let $\mathcal{C}$ be **Cpo** (directed-complete posets and continuous maps). Let a $\mathcal{D}$-object be a pointed cpo (a cpo with a least element) 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 $\underline{Z}$. 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 judgement model $(\mathcal{C}, \mathcal{C}^T, \mathcal{O}^T)$, 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 $\mathsf{self}\,\mathbf{Set}$ and let $\mathcal{O}_X Y$ be $\mathbf{Set}(S \times X, Y)$.

erratic choice Let $\mathcal{C}$ be **Set**, let $\mathcal{D}$ be the locally **Set**-indexed category **Rel** in which 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 **Set**, let $\mathcal{D}$ be $(\mathsf{self}\,\mathbf{Set})^{\mathsf{op}}$ and let $\mathcal{O}_X Y$ be $\mathbf{Set}(X \times Y, R)$.

4.10. EXAMPLE: POSSIBLE WORLDS. Our next example is a possible world semantics. This is a way of modelling dynamically generated storage cells that was introduced for CBN in [Ole82] and for CBV in [Mog90]. These were models of ground storage only, but a CBV model for general storage was presented in [Lev02], and our example is the CBPV version of it.

Let $\mathcal{W}$ be a countable category (countably many objects and morphisms), and for each $w \in \mathsf{ob}\,\mathcal{W}$ let Sw be a set. (The storage model above is the special case where $\mathcal{W}$ is the unit category.) We define $\mathcal{C}$ to be $[\mathcal{W}, \mathbf{Set}]$, and $\mathcal{D}$ to be the locally $\mathcal{C}$-indexed category where

- an object is a functor from $\mathcal{W}^{\mathsf{op}}$ to **Set**
- a morphism over X from $\underline{Y}$ to $\underline{Z}$ is a dinatural transformation from $X \times \underline{Y}$ to $\underline{Z}$.

We define $\mathcal{O}_X\underline{Y}$ to be

$$\prod_{w \in \mathsf{ob}\ \mathcal{W}} \mathbf{Set}(Sw \times Xw, \underline{Y}w)$$

(Thus there is no naturality condition on a computation-morphism, nor can there be, because S is not a functor.) Identity, composition and reindexing are defined in the evident way.

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.

5.2. Definition. In a CBPV judgement model $(\mathcal{C}, \mathcal{D}, \mathcal{O})$,

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

$$\begin{array}{rclr} \mathcal{C}(X, V) & \longrightarrow & \mathcal{O}_X\underline{B} & \text{for all } X \\ f & \longmapsto & f^*\mathsf{force} & \end{array} \tag{9}$$

are isomorphisms

$\boldsymbol{FA}$ a *left adjunctive* for a $\mathcal{C}$-object A is a $\mathcal{D}$-object $\underline{V}$ and an $\mathcal{O}$-morphism $\xrightarrow[A]{\mathsf{return}} \underline{V}$, such that the functions

$$\begin{array}{rclr} \mathcal{D}_X(\underline{V}, \underline{Y}) & \longrightarrow & \mathcal{O}_{X\times A}\underline{Y} & \text{for all } X, \underline{Y} \\ h & \longmapsto & (\pi'^*_{X,A}\mathsf{return}); (\pi^*_{X,A}h) & \end{array} \tag{10}$$

are isomorphisms

$\sum_{i \in I} \boldsymbol{A_i}$ a *distributive 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

$$\begin{array}{rclrr} \mathcal{C}(X \times V, Y) & \longrightarrow & \prod_{i\in I}\mathcal{C}(X \times A_i, Y) & \text{for all } X, Y & (11) \\ \mathcal{O}_{X\times V}\underline{Y} & \longrightarrow & \prod_{i\in I}\mathcal{O}_{X\times A_i}\underline{Y} & \text{for all } X, \underline{Y} & (12) \\ \mathcal{D}_{X\times V}(\underline{Y}, \underline{Z}) & \longrightarrow & \prod_{i\in I}\mathcal{D}_{X\times A_i}(\underline{Y}, \underline{Z}) & \text{for all } X, \underline{Y}, \underline{Z} & (13) \\ f & \longmapsto & \lambda i.((X \times \mathsf{in}_i)^* f) & & \end{array}$$

are isomorphisms

$\prod_{i \in I} \underline{\boldsymbol{B}_i}$ 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

$$\begin{array}{rcll} \mathcal{O}_X\underline{V} & \longrightarrow & \prod_{i\in I}\mathcal{O}_X\underline{B}_i & \text{for all } X \quad (14) \\ \mathcal{D}_X(\underline{Y},\underline{V}) & \longrightarrow & \prod_{i\in I}\mathcal{D}_X(\underline{Y},\underline{B}_i) & \text{for all } X,\underline{Y} \quad (15) \\ h & \longmapsto & \lambda i.(h;(\,)^*\pi_i) & \end{array}$$

are isomorphisms

$\boldsymbol{A} \rightarrow \underline{\boldsymbol{B}}$ 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

$$\begin{array}{rcll} \mathcal{O}_X\underline{V} & \longrightarrow & \mathcal{O}_{X\times A}\underline{B} & \text{for all } X \quad (16) \\ \mathcal{D}_X(\underline{Y},\underline{V}) & \longrightarrow & \mathcal{D}_{X\times A}(\underline{Y},\underline{B}) & \text{for all } X,\underline{Y} \quad (17) \\ h & \longmapsto & (\pi^*_{X,A}h);(\pi'^*_{X,A}\mathsf{ev}) & \end{array}$$

are isomorphisms.

5.3. Definition. A *strong adjunction* is a CBPV judgement model $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ 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

5.4. Proposition. [Lev01] A strong adjunction from $\mathcal{C}$ to $\mathcal{D}$ is equivalent to an adjunction from $\mathsf{self}\,\mathcal{C}$ to $\mathcal{D}$.

Therefore a strong adjunction from $\mathcal{C}$ gives rise to a monad on $\mathsf{self}\,\mathcal{C}$ i.e. (by Prop. 4.4) a strong monad on $\mathcal{C}$; hence the word "strong".

5.5. Definition. A *CBPV adjunction* is a strong adjunction $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ with all countable distributive coproducts, countable products and exponentials. We write U,F,$\rightarrow$ etc. for the operations on objects. We write q^U for the inverse of (9), etc.

It is obvious how to define the interpretation of CBPV in such a structure. The following result shows that there is some redundancy in the definition of CBPV adjunction.

5.6. Proposition. In a CBPV judgement model with all left adjunctives,

products if the functions (15) are all isomorphisms then so are the functions (14)

exponentials if the functions (17) are all isomorphisms then so are the functions (16).

In a CBPV judgement model with all right adjunctives,

distributive coproducts if the functions (11) are all isomorphisms then so are the functions (12).

Proof.

products The isomorphism

$$\mathcal{D}_X(F1,\underline{Y}) \xrightarrow{\cong} \mathcal{O}_{X\times 1}\underline{Y} \xrightarrow{\cong} \mathcal{O}_X\underline{Y}$$

maps h to $\mathsf{return};h$, by calculation. So the diagram

$$\begin{tikzcd}
\mathcal{D}_X(F1,\underline{V}) \arrow[r] \arrow[d, "\cong"'] & \prod_{i\in I}\mathcal{D}_X(F1,\underline{B}_i) \arrow[d, "\cong"] \\
\mathcal{O}_X(F1,\underline{V}) \arrow[r] & \prod_{i\in I}\mathcal{O}_X\underline{B}_i
\end{tikzcd}$$

commutes, by calculation. So if the top arrow is an isomorphism, the bottom arrow must be too.

exponentials Similar, using the diagram

$$\begin{tikzcd}
\mathcal{D}_X(F1,\underline{V}) \arrow[r] \arrow[d, "\cong"'] & \mathcal{D}_{X\times A}(F1,\underline{B}) \arrow[d, "\cong"] \\
\mathcal{O}_X\underline{V} \arrow[r] & \mathcal{O}_{X\times A}\underline{B}
\end{tikzcd}$$

distributive coproducts The diagram

$$\begin{tikzcd}
\mathcal{C}_{X\times V}U\underline{Y} \arrow[r] \arrow[d, "\cong"'] & \prod_{i\in I}\mathcal{C}_{X\times A_i}U\underline{Y} \arrow[d, "\cong"] \\
\mathcal{O}_{X\times V}\underline{Y} \arrow[r] & \prod_{i\in I}\mathcal{O}_{X\times A_i}\underline{Y}
\end{tikzcd}$$

commutes by calculation. So if the top arrow is an isomorphism, the bottom arrow must be too.

■

We draw attention to the requirement for (13) to be an isomorphism, which is not eliminated by Prop. 5.6. It was mistakenly omitted in [Lev01] (which merely required $\mathcal{C}$ to be countably distributive), and was brought to light by the stack judgement.

5.7. Examples. We now look again at the examples of CBPV judgement models described in Sect. 4.9 to see when they have all the required universal elements. Notice how the strong monad described in [Mog91] is recovered in all cases treated there. For the possible world model, where the strong monad mentioned in [Lev02] is recovered, and for the game model, the strong monad described in [AM98] is recovered.

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

cpo This is clearly a CBPV adjunction with FA given as the lift of A, and $U\underline{B} = \underline{B}$.

monad/algebra The judgement model given by a strong monad T on a cartesian category $\mathcal{C}$ is a CBPV adjunction 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.

where FA is the free T-algebra on A and $U\underline{B}$ is the carrier of $\underline{B}$. Notice that these conditions imply that $\mathcal{C}$ has all Kleisli exponentials and all countable products of Kleisli exponentials.

storage, erratic choice, continuations The connectives are given by

model	$\sum_{i\in I}$	1	$\times$	U	F	$\prod_{i\in I}$	$\to$
storage	$\sum_{i\in I}$	1	$\times$	$S \to -$	$S \times -$	$\prod_{i\in I}$	$\to$
erratic choice	$\sum_{i\in I}$	1	$\times$	$\mathcal{P}$	$-$	$\sum_{i\in I}$	$\times$
continuations	$\sum_{i\in I}$	1	$\times$	$- \to R$	$- \to R$	$\sum_{i\in I}$	$\times$

possible worlds The $\sum 1 \times \prod \to$ connectives are given pointwise. For example, the exponential $A \to \underline{B}$ is given at w by $Aw \to \underline{B}w$, which must be contravariant in w because A is covariant and $\underline{B}$ is contravariant. For the adjunctives, recall the *coslice category* $w/\mathcal{W}$, in which an object is a world w' together with a morphism $w \xrightarrow{g} w'$. The adjunctives are given at a world w by

$$\begin{aligned} (FA)w &= \sum_{(w',g)\in w/\mathcal{W}} (Sw' \times Aw') \\ (U\underline{B})w &= \prod_{(w',g)\in w/\mathcal{W}} (Sw' \to \underline{B}w') \end{aligned}$$

and at a morphism $x \xrightarrow{f} w$ by

$$\begin{aligned} ((FA)f)(w',\, w \xrightarrow{g} w', s', a) &= (w', (f;g), s', a) \\ (((U\underline{B})f)\alpha)(w',\, w \xrightarrow{g} w', s') &= \alpha(w', (f;g), s') \end{aligned}$$

5.8. General Storage Models. We can generalize some of the examples in Sect. 5.7 (global store, possible worlds, continuations) into ways of building CBPV models from other CBPV models. This enables us to model combinations of effects, rather like the monad transformers of [CM93].

For global store, suppose that S is a $\mathcal{C}$-object in a CBPV adjunction $(\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$. Then we obtain a storage model $(\mathcal{C}, \mathcal{D}, \mathcal{O}')$ where $\mathcal{O}'_X \underline{Y}$ is defined to be $\mathcal{O}_{S \times X} \underline{Y}$. The $\sum 1 \times \prod \rightarrow$ connectives are unchanged, and the adjunctives are given by

$$\begin{array}{rcl} F'A & = & F(S \times A) \\ U'\underline{B} & = & U(S \to \underline{B}) \end{array}$$

For possible worlds, suppose we are given a CBPV adjunction $(\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$, a category $\mathcal{W}$ with countably many objects and morphisms, and a $\mathcal{C}$-object Sw for each $w \in \mathcal{W}$. We then obtain a possible world model $(\mathcal{C}', \mathcal{D}', \mathcal{O}')$ as follows.

- $\mathcal{C}'$ is defined to be $[\mathcal{W}, \mathcal{C}]$
- a $\mathcal{D}'$-object is a functor from $\mathcal{W}^{\mathsf{op}}$ to $\mathcal{D}_1$
- a $\mathcal{D}'$ morphism $\underline{Y} \xrightarrow[X]{h} \underline{Z}$ provides for each world w a morphism $\underline{Y}w \xrightarrow[Xw]{hw} \underline{Z}w$, such that the diagram over Xw

$$\begin{tikzcd} \underline{Y}w \arrow[r, "hw"] & \underline{Z}w \\ \underline{Y}w' \arrow[u, "()^*\underline{Y}f"] \arrow[r, "(Xf)^*(hw')"'] & \underline{Z}w' \arrow[u, "()^*\underline{Z}f"'] \end{tikzcd}$$

  commutes for each $w \xrightarrow{f} w'$
- $\mathcal{O}'_X \underline{Y}$ is defined to be

$$\prod_{w \in \mathcal{W}} \mathcal{O}_{Sw \times Xw} \underline{Y}w$$

- identity, composition and reindexing are defined in the evident way
- the $\sum 1 \times \prod \rightarrow$ connectives are given pointwise
- the adjunctives are given at a world w by

$$\begin{array}{rcl} (FA)w & = & F \sum_{(w',g) \in w/\mathcal{W}} (Sw' \times Aw') \\ (U\underline{B})w & = & U \prod_{(w',g) \in w/\mathcal{W}} (Sw' \to \underline{B}w') \end{array}$$

  and at a morphism $x \xrightarrow{f} w$ in the evident way.

Again, the global store construction is the special case that $\mathcal{W}$ is the unit category.

5.9. General Continuation Models. To see the general construction of continuation models, it is helpful to introduce a calculus. Let us write CBPV+$\underline{\mathtt{R}}$ for CBPV extended with a free computation type identifier $\underline{\mathtt{R}}$. This has a special fragment called *Jump-With-Argument* (JWA), shown in Fig. 4–5. (Many similar continuation calculi appear in the literature [Dan92, LRS93, SF93, Sel01, Thi97].) JWA contains two kinds of terms: values and *nonreturning commands*. We write $\Gamma \vdash^{\mathsf{n}} M$ to say that M is a nonreturning command in context Γ.

Types

$$A ::= \neg A \mid \textstyle\sum_{i\in I} A_i \mid 1 \mid A \times A$$

Judgements

$$\Gamma \vdash^{\mathsf{v}} V : A \qquad\qquad \Gamma \vdash^{\mathsf{n}} M$$

Terms:

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

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

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

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{v}} \gamma\mathtt{x}.M : \neg A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} W : \neg A}{\Gamma \vdash^{\mathsf{n}} V \nearrow W}$$

Figure 4: Syntax of Jump-With-Argument

$$\begin{array}{rcl}
\neg A & \text{is} & U(A \to \underline{\mathtt{R}}) \\
\Gamma \vdash^{\mathsf{n}} M & \text{is} & \Gamma \vdash^{\mathsf{c}} M : \underline{\mathtt{R}} \\
\gamma\mathtt{x}.M & \text{is} & \mathtt{thunk}\ \lambda\mathtt{x}.M \\
V \nearrow W & \text{is} & V\text{`}\mathtt{force}\ W
\end{array}$$

Figure 5: JWA as fragment of CBPV+$\underline{\mathtt{R}}$

We can substitute a context morphism $\Gamma \xrightarrow{q} \Delta$ into a command $\Delta \vdash^{\mathsf{n}} M$, giving a command $\Gamma \vdash^{\mathsf{n}} q^*M$. This operation satisfies

$$\begin{array}{rcl} \mathsf{id}^*M & = & M \\ p^*q^*M & = & (p;q)^*M \end{array}$$

For JWA, as for CBPV and CBV, we require a cartesian category to model the values. For nonreturning commands, we require a functor $\mathcal{N} : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Set}$, also called a left $\mathcal{C}$-module. This can be describe in concrete terms.

5.10. Definition.

1. A *left $\mathcal{C}$-module $\mathcal{N}$* consists of

   - for each $A \in \mathsf{ob}\ \mathcal{C}$, a set $\mathcal{N}_A$ of *$\mathcal{N}$-morphisms from A*, written $A \xrightarrow{g}$
   - for each $\mathcal{C}$-morphism $A \xrightarrow{f} A'$ and $\mathcal{N}$-morphism $A' \xrightarrow{g}$, a *composite* $\mathcal{N}$-morphism $A \xrightarrow{f;g}$

   satisfying identity and associativity laws

$$\begin{array}{rcl} \mathsf{id};g & = & g \\ (f;f');g & = & f;(f';g) \end{array}$$

   where g is a $\mathcal{N}$-morphism.

2. A *JWA judgement model* is a cartesian category $\mathcal{C}$ together with a left $\mathcal{C}$-module $\mathcal{N}$.

The only connectives we can interpret in a JWA judgement model are 1 and $\times$. To interpret the other connectives, we once again need appropriate universal properties.

5.11. Definition. In a JWA judgement model $(\mathcal{C}, \mathcal{N})$,

$\neg \boldsymbol{A}$ a *jumpwith* for a $\mathcal{C}$-object A is a $\mathcal{C}$-object V and a $\mathcal{N}$-morphism $V \times A \xrightarrow{\mathsf{jump}}$ such that the functions

$$\begin{array}{rcl} \mathcal{C}_X V & \longrightarrow & \mathcal{N}_{X \times A} \qquad \text{for all } X \\ f & \longmapsto & (f \times A)^*\mathsf{jump} \end{array} \tag{18}$$

are isomorphisms.

$\sum_{\boldsymbol{i \in I}} \boldsymbol{A_i}$ a *distributive 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

$$\mathcal{C}(X \times V, Y) \longrightarrow \prod_{i \in I} \mathcal{C}(X \times A_i, Y) \qquad \text{for all } X, Y \tag{19}$$

$$\begin{array}{rcl} \mathcal{N}_{X \times V} & \longrightarrow & \prod_{i \in I} \mathcal{N}_{X \times A_i} \qquad \text{for all } X \\ f & \longmapsto & \lambda i.((X \times \mathsf{in}_i)^* f) \end{array} \tag{20}$$

are isomorphisms.

A *JWA model* is a JWA judgement model with all jumpwiths and all countable distributive coproducts, and we write $\neg$ and $\sum$ for the operations on objects.

There is some redundancy in the definition of JWA model, by analogy with Prop. 5.6.

5.12. PROPOSITION. In a JWA judgement model with all jumpwiths,

distributive coproducts if the functions (19) are all isomorphisms then so are the functions (20).

PROOF. The isomorphism

$$\mathcal{C}_X \neg 1 \xrightarrow{\cong} \mathcal{N}_{X\times 1} \xrightarrow{\cong} \mathcal{N}_X$$

maps f to $(f, ())^* \mathsf{jump}$, by calculation. So the diagram

$$\begin{tikzcd}
\mathcal{C}_{X\times V}\,\neg 1 \arrow[r] \arrow[d, "\cong"'] & \prod_{i\in I} \mathcal{C}_{X\times A_i}\,\neg 1 \arrow[d, "\cong"] \\
\mathcal{N}_{X\times V} \arrow[r] & \prod_{i\in I} \mathcal{N}_{X\times A_i}
\end{tikzcd}$$

commutes by calculation. So if the top arrow is an isomorphism, the bottom must be too. ∎

Now we know what a JWA model is, we can define 2 constructions.

1. Given a CBPV model $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ and $\mathcal{D}$-object $\underline{R}$, we obtain a JWA model $(\mathcal{C}, \mathcal{N})$ by setting $\mathcal{N}_X = \mathcal{O}_X \underline{R}$. The $\sum 1\times$ connectives are unchanged, and jumpwiths are given by $\neg A = U(A \to \underline{R})$.

2. Suppose we are given a JWA model $(\mathcal{C}, \mathcal{N})$. Then we obtain a CBPV adjunction $(\mathcal{C}, \mathcal{D}', \mathcal{O}')$ where $\mathcal{D}'$ is $(\mathsf{self}\,\mathcal{C})^{\mathsf{op}}$ and $\mathcal{O}'_X Y$ is defined to be $\mathcal{N}_{X\times Y}$. The $\sum 1\times$ connectives are unchanged, and the other connectives are given by

$$\begin{array}{rcl}
F'A & = & \neg A \\
U'B & = & \neg B \\
A \to' B & = & A \times B \\
\prod'_{i\in I} B_i & = & \sum_{i\in I} B_i
\end{array}$$

6. Game Models

We wish to show the game semantics of [AHM98] to be a CBPV adjunction. It is convenient to do this first *without* the bracketing condition, in the manner of [Lai97], which is easier because it is a continuation model. Then we adapt the definitions to incorporate the bracketing condition. It is also possible to treat models constrained by visibility and innocence in the manner of [HO00, AM98], but we do not describe this here.

6.1. JWA Model Without Bracketing Condition. We first construct the JWA model. The basic definitions are standard:

6.2. Definition.

1. An (unlabelled) *arena* is a countable forest, i.e. a countable set R of *tokens* together with a function $\mathsf{enabler} : R \longrightarrow \{*\} + R$ such that, for every token r, the sequence

$$r = r_0 \xmapsto{\mathsf{enabler}} r_1 \xmapsto{\mathsf{enabler}} \cdots$$

   eventually reaches $*$. We usually write R instead of $(R, \mathsf{enabler})$, and we write $\mathsf{rt}\ R$ for the set of tokens enabled by $*$.

2. A *play* in an arena R is a sequence of tokens $t_0, \ldots, t_{n-1}$, together with a function $\mathsf{justifier} : \{0, \ldots, n-1\} \longrightarrow \{*, 0, \ldots, n-1\}$ such that, for all $i \in \{0, \ldots, n-1\}$, we have

   - $\mathsf{justifier}\ i < i$
   - $\mathsf{enabler}\ t_i = t_{\mathsf{justifier}\ i}$
   - $\mathsf{justifier}\ i$ is even iff i is odd

   where $*$ is taken to be odd and < 0, and t_* is defined to be $*$.

By contrast with the usual accounts, we make Player go first. So, for a play of length n, we call $i \in \{0, \ldots, n-1\}$ a P-*move* if i is even and an O-*move* if i is odd. Similarly, we say the play is *awaiting* P if n is even, and *awaiting* O if n is odd.

6.3. Definition. A *strategy* for an arena R is a prefix-closed set σ of O-awaiting plays that is deterministic i.e. if $sm \in \sigma$ and $sn \in \sigma$ then $m = n$. We write $\mathsf{strat}\ R$ for the set of strategies on R.

This enables us to describe the homsets of the "thread-independent" category $\mathcal{G}$ of [AHM98] (without the bracketing condition) by

$$\mathcal{G}(R, S) = \prod_{s \in \mathsf{rt}\ S} \mathsf{strat}(R \uplus S\restriction_s) \tag{21}$$

where we write $\uplus$ for disjoint union of arenas, and $S\restriction_s$ for the arena of tokens in S strictly below s.

We then define our JWA model $(\mathcal{C}, \mathcal{N})$ in the manner of [AM98]. Thus, a $\mathcal{C}$-object is a countable family of arenas, and the homsets are given by

$$\begin{aligned} \mathcal{C}(\{R_i\}_{i \in I}, \{S_j\}_{j \in J}) &= \prod_{i \in I} \sum_{j \in J} \mathcal{G}(R_i, S_j) \\ \mathcal{N}_{\{R_i\}_{i \in I}} &= \prod_{i \in I} \mathsf{strat}\ R_i \end{aligned}$$

Identities and composition are defined in the usual way. The connectives are given as follows:

$$\begin{array}{rcl}
\{R_i\}_{i\in I}\times\{S_j\}_{j\in J} & = & \{R_i\uplus S_j\}_{(i,j)\in I\times J} \\
\sum_{i\in I}\{R_{ij}\}_{j\in J_i} & = & \{R_{ij}\}_{(i,j)\in\sum_{i\in I}J_i} \\
\neg\{R_i\}_{i\in I} & = & \{\mathsf{pt}_{i\in I}R_i\}_{()\in 1}
\end{array}$$

where $\mathsf{pt}_{i\in I}R_i$ is the arena with I roots, and a copy of R_i pasted beneath the ith root.

6.4. CBPV Adjunction Model Without Bracketing Condition. We next apply the construction of Sect. 5.9 to obtain a CBPV model $(\mathcal{C},\mathcal{D}',\mathcal{O}')$ from our JWA model. Let us write this out explicitly. An object of $\mathcal{D}$, like an object of $\mathcal{C}$, is a countable family of arenas. The homsets are described by

$$\begin{array}{rcl}
\mathcal{C}(\{R_i\}_{i\in I},\{S_j\}_{j\in J}) & = & \prod_{i\in I}\sum_{j\in J}\mathcal{G}(R_i,S_j) \\
\mathcal{O}'_{\{R_i\}_{i\in I}}\{S_j\}_{j\in J} & = & \prod_{i\in I}\prod_{j\in J}\mathsf{strat}(R_i\uplus S_j) \\
\mathcal{D}'_{\{R_i\}_{i\in I}}(\{S_j\}_{j\in J},\{T_k\}_{k\in K}) & = & \prod_{i\in I}\prod_{k\in K}\sum_{j\in J}\mathcal{G}(R_i\uplus T_k,S_j)
\end{array}$$

The connectives are described by

$$\begin{array}{rcl}
\{R_i\}_{i\in I}\times\{S_j\}_{j\in J} & = & \{R_i\uplus S_j\}_{(i,j)\in I\times J} \\
\sum_{i\in I}\{R_{ij}\}_{j\in J_i} & = & \{R_{ij}\}_{(i,j)\in\sum_{i\in I}J_i} \\
U\{R_i\}_{i\in I} & = & \{\mathsf{pt}_{i\in I}R_i\}_{()\in 1} \\
F\{R_i\}_{i\in I} & = & \{\mathsf{pt}_{i\in I}R_i\}_{()\in 1} \\
\prod_{i\in I}\{R_{ij}\}_{j\in J_i} & = & \{R_{ij}\}_{(i,j)\in\sum_{i\in I}J_i} \\
\{R_i\}_{i\in I}\to\{S_j\}_{j\in J} & = & \{R_i\uplus S_j\}_{(i,j)\in I\times J}
\end{array}$$

6.5. CBPV Adjunction Model With Bracketing Condition. To describe the bracketing condition, we first need to classify tokens into "question" and "answer" tokens.

6.6. Definition.

1. A (Q/A-labelled) *arena* is an unlabelled arena $(R,\mathsf{enabler})$ together with a labelling function $R\xrightarrow{\lambda^{\mathsf{QA}}}\{\mathsf{Q},\mathsf{A}\}$ such that every token enabled by an answer is a question. We say the arena is Q-*rooted* when, moreover, every root is a question.
2. For a countable family of arenas $\{R_i\}_{i\in I}$, we write $\mathsf{pt}^{\mathsf{Q}}_{i\in I}R_i$ for the arena with I roots, all labelled Q, and a copy of R_i placed below the ith root.
3. For a countable family of Q-rooted arenas $\{R_i\}_{i\in I}$, we write $\mathsf{pt}^{\mathsf{A}}_{i\in I}R_i$ for the arena with I roots, all labelled A, and a copy of R_i placed below the ith root.

6.7. DEFINITION.

1. Let s be a P-awaiting play. Its *pending question* is the O-move Q_0 if s is of the form

$$\cdots \quad Q_0 \quad [\ Q \cdots A\ ]^*$$

and $*$ if s is of the form

$$[\ Q \cdots A\ ]^*$$

2. Let s be an O-awaiting play. Its *pending question* is the P-move Q_0 if s is of the form

$$\cdots \quad Q_0 \quad [\ Q \cdots A\ ]^*$$

and does not exist if s is of the form

$$\cdots \quad A_0 \quad [\ Q \cdots A\ ]^*$$

where A_0 is justified by $*$.

3. A play is *well-bracketed* when, for every prefix of the form sm where m is an A-move, m points to the pending question of s.

4. A *strategy under the bracketing condition* on an arena R is a prefix-closed set of O-awaiting well-bracketed plays on R satisfying determinism: if $sm = \sigma$ and $sn = \sigma$ then $m = n$. We write $\mathsf{strat}^{\mathsf{b}}\ R$ for the set of all strategies under the bracketing condition on R.

By analogy with (21), we define

$$\mathcal{G}^{\mathsf{b}}(R, S) = \prod_{s \in \mathsf{rt}\ S} \mathsf{strat}^{\mathsf{b}}(R \uplus S\restriction_s)$$

Now we construct our CBPV adjunction $(\mathcal{C}, \mathcal{D}', \mathcal{O}')$. A $\mathcal{C}$-object is a family of Q-rooted arenas, whereas a $\mathcal{D}'$-object is a family of arenas (not necessarily Q-rooted). The adjunctives are given by

$$\begin{array}{rcl} U\{R_i\}_{i \in I} & = & \{\mathsf{pt}^{\mathsf{Q}}_{i \in I} R_i\}_{() \in 1} \\ F\{R_i\}_{i \in I} & = & \{\mathsf{pt}^{\mathsf{A}}_{i \in I} R_i\}_{() \in 1} \end{array}$$

Everything else is as in Sect. 6.4, replacing strat by $\mathsf{strat}^{\mathsf{b}}$ and $\mathcal{G}$ by $\mathcal{G}^{\mathsf{b}}$. All the (omitted) definitions of identity, reindexing and composition adapt to the well-bracketed setting. The value category and strong monad obtained are precisely those defined in [AHM98].

6.8. REMARK. In [AHM98], the bracketing condition is imposed on both players, and the categorical structure used there is dependent on that. The formulation given here has the advantage that it works whether bracketing is imposed on both players, on neither player, or on Player alone (as in [Har99]).

7. The Equational Theory

7.1. Complex Values and Stacks. Since we want the term model of CBPV+stacks to be a CBPV adjunction, 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. 6. We shall see in Sect. 7.2 that these new constructs can always be eliminated from a computation. For example, $\mathtt{return}\ (\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W)$ can be simplified into $\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\mathtt{return}\ W$.

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. 8.2; they are only a convenience. The pattern-match constructs, on the other hand, are essential.

7.2. Properties of Equational Theory. The equational theory is the least congruence on terms-in-context containing the laws in Fig. 7. To reduce clutter, we omit the assumptions necessary to make each equation well-typed. Given a term P in context Γ, we explicitly write its weakening by $\mathtt{x} : A$ as ${}^{\mathtt{x}:A}P$ or just ${}^{\mathtt{x}}P$. Since a context must consist of distinct identifiers, this notation implicitly assumes that $\mathtt{x} \notin \Gamma$, so we avoid the need for the traditional proviso $\mathtt{x} \notin \mathsf{FV}(P)$.

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

Proof. Straightforward induction in each case. ∎

7.4. Proposition. 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 Fig. 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. 7.4, which does not involve stacks, is proved in [Lev01].

One noteworthy equation of Fig. 7 is the η-law

$$M \bullet K = M\ \mathtt{to}\ \mathtt{x}.\ ((\mathtt{return}\ \mathtt{x}) \bullet {}^{\mathtt{x}}K) \tag{22}$$

This is equivalent to the two equations

$$\begin{aligned} M &= M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{return}\ \mathtt{x} && (23)\\ (M\ \mathtt{to}\ \mathtt{x}.\ N) \bullet K &= M\ \mathtt{to}\ \mathtt{x}.\ (N \bullet {}^{\mathtt{x}}K) && (24) \end{aligned}$$

Complex Values

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

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

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

Complex Stacks

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

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

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

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

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

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

Figure 6: Complex Values and Stacks

(24) makes it clear why a stack denotes an algebra homomorphism in the monad models of CBPV. As instances of (24) we have many familiar CBPV equations:

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

In [Lev01] rather complex lemmas were required to prove these valid in an adjunction semantics. It is an advantage of working with stacks that these equations become straightforward.

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

$$\begin{array}{ccc} & \boldsymbol{\eta}\textbf{-laws} & \\ Q[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).\,{}^{\mathtt{x}}Q[(i,\mathtt{x})/\mathtt{z}],\ldots\} \\ Q[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\,{}^{\mathtt{xy}}Q[(\mathtt{x},\mathtt{y})/\mathtt{z}] \\ V & = & \mathtt{thunk}\ \mathtt{force}\ V \\ M\bullet K & = & M\ \mathtt{to}\ \mathtt{x}.\ ((\mathtt{return}\ \mathtt{x})\bullet {}^{\mathtt{x}}K) \\ K \mathbin{+\!\!+} L & = & [\cdot]\ \mathtt{to}\ \mathtt{x}.\ ((\mathtt{return}\ \mathtt{x})\bullet {}^{\mathtt{x}}K)::L \\ M & = & \lambda\{\ldots,i.i`M,\ldots\} \\ K \mathbin{+\!\!+} L & = & \{\ldots,(K \mathbin{+\!\!+} i::\mathtt{nil})\ \mathtt{where}\ i::\mathtt{nil},\ldots\}\ \mathtt{is}\ L \\ M & = & \lambda\mathtt{x}.(\mathtt{x}`\,{}^{\mathtt{x}}M) \\ K \mathbin{+\!\!+} L & = & ({}^{\mathtt{x}}K \mathbin{+\!\!+} \mathtt{x}::\mathtt{nil})\ \mathtt{where}\ \mathtt{x}::\mathtt{nil}\ \mathtt{is}\ L \end{array}$$

Figure 7: Equational laws for CBPV + stacks

8. General Theories

8.1. 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 [LS86], we need the facility to add primitive operations to CBPV. Each operation has a "sorting" which is a sequent[2]. For example, the sorting $+$ is $\mathtt{nat},\mathtt{nat} \vdash^{\mathsf{v}} \mathtt{nat}$. Applied to two values of type $\mathtt{nat}$, it makes a value of type $\mathtt{nat}$. An operation $\mathtt{div}$ that returns a natural number, or—if the divisor is zero—raises an error, has the sorting $\mathtt{nat},\mathtt{nat} \vdash^{\mathsf{c}} F\mathtt{nat}$.

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

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

A *signature* is a collection of primitive operations; more formally, a function from sequents to sets. The associated rules are shown in Fig. 8, and in Fig. 9 we define dismantling and concatenation for the terms generated by a signature.

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

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

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

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

Figure 8: Rules For A Signature S

8.2. 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 [LS86], 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.

[2]In [LS86], 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.

K	$M \bullet K$	$K \mathbin{+\!\!+} L$
$\mathtt{nil}$	M	L
$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ N :: K$	$(M\ \mathtt{to}\ \mathtt{x}.\ N) \bullet K$	$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ N :: (K \mathbin{+\!\!+} L)$
$\hat{\imath} :: K$	$(\hat{\imath}\text{`}M) \bullet K$	$\hat{\imath} :: (K \mathbin{+\!\!+} L)$
$V :: K$	$(V\text{`}M) \bullet K$	$V :: (K \mathbin{+\!\!+} L)$
$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ K$	$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ ({}^{\mathtt{x}}M \bullet K)$	$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ (K \mathbin{+\!\!+} {}^{\mathtt{x}}L)$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).K_i,\ldots\}$	$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).({}^{\mathtt{x}}M \bullet K_i),\ldots\}$	$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).K_i \mathbin{+\!\!+} {}^{\mathtt{x}}L,\ldots\}$
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).K$	$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).({}^{\mathtt{xy}}M \bullet K)$	$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).(K \mathbin{+\!\!+} {}^{\mathtt{xy}}L)$
$K\ \mathtt{where}\ \mathtt{nil}\ \mathtt{is}\ K'$	$(M \bullet K) \bullet K'$	$K\ \mathtt{where}\ \mathtt{nil}\ \mathtt{is}\ (K' \mathbin{+\!\!+} L)$
$\{\ldots,K_i\ \mathtt{where}\ i :: \mathtt{nil},\ldots\}\ \mathtt{is}\ K'$	$(\lambda\{\ldots,i.(M \bullet K_i),\ldots\}) \bullet K'$	$\{\ldots,K_i\ \mathtt{where}\ i :: \mathtt{nil},\ldots\}\ \mathtt{is}\ (K' \mathbin{+\!\!+} L)$
$K\ \mathtt{where}\ \mathtt{x} :: \mathtt{nil}\ \mathtt{is}\ K'$	$(\lambda\mathtt{x}.(M \bullet K)) \bullet K'$	$K\ \mathtt{where}\ \mathtt{x} :: \mathtt{nil}\ \mathtt{is}\ (K' \mathbin{+\!\!+} L)$
$f(V_0,\ldots,V_{r-1}\|[\cdot]) :: K$	$f(V_0,\ldots,V_{r-1}\|M) \bullet K$	$f(V_0,\ldots,V_{r-1}\|[\cdot]) :: (K \mathbin{+\!\!+} L)$

Figure 9: Dismantling and Concatenation

P	$[\![P]\!]$
$\mathtt{x}_i$	π_i
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ P$	$(\mathsf{id}, [\![V]\!])^*[\![P]\!]$
$\mathtt{return}\ V$	$[\![V]\!]^*\mathsf{prod}$
$M\ \mathtt{to}\ \mathtt{x}.\ N$	$[\![M]\!]; \mathsf{q}^F[\![N]\!]$
$\mathtt{force}\ V$	$[\![V]\!]^*\mathsf{force}$
$\mathtt{thunk}\ M$	$\mathsf{q}^U[\![M]\!]$
$\lambda\mathtt{x}.M$	$\mathsf{q}^{\rightarrow}[\![M]\!]$
$V`M$	$[\![M]\!]; [\![V]\!]^*\mathsf{ev}$
$\lambda\{\ldots, i.M_i, \ldots\}$	$\mathsf{q}^{\Pi}\lambda i.[\![M_i]\!]$
$\hat{\imath}`M$	$[\![M]\!]; (\,)^*\pi_{\hat{\imath}}$
(V, V')	$([\![V]\!], [\![V']\!])$
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).\ P$	$((\mathsf{id}, ([\![V]\!]; \pi)), ([\![V']\!]; \pi'))^*[\![P]\!]$
$(\hat{\imath}, V)$	$[\![V]\!]; \mathsf{in}_{\hat{\imath}}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).P_i, \ldots\}$	$(\mathsf{id}, [\![V]\!])^*\mathsf{q}^{\Sigma}\lambda i.[\![P_i]\!]$
$\mathtt{nil}$	id
$[\cdot]\ \mathtt{to}\ \mathtt{x}.\ M :: K$	$(\mathsf{q}^F[\![M]\!]); [\![K]\!]$
$\hat{\imath} :: K$	$((\,)^*\pi_{\hat{\imath}}); [\![K]\!]$
$V :: K$	$([\![V]\!]^*\mathsf{ev}); [\![K]\!]$
$K\ \mathtt{where}\ \mathtt{nil}\ \mathtt{is}\ L$	$[\![K]\!]; [\![L]\!]$
$\{\ldots, K_i\ \mathtt{where}\ i :: \mathtt{nil}, \ldots\}\ \mathtt{is}\ L$	$(\mathsf{q}^{\Pi}\lambda i.[\![K_i]\!]); [\![L]\!]$
$K\ \mathtt{where}\ \mathtt{x} :: \mathtt{nil}\ \mathtt{is}\ L$	$(\mathsf{q}^{\rightarrow}[\![K]\!]); [\![L]\!]$
$f(V_0, \ldots, V_{r-1})$	$([\![V_0]\!], \ldots, [\![V_{r-1}]\!])^*f$
$f(V_0, \ldots, V_{r-1}\|M)$	$[\![M]\!]; ([\![V_0]\!], \ldots, [\![V_{r-1}]\!])^*f$
$f(V_0, \ldots, V_{r-1}\|[\cdot]) :: K$	$([\![V_0]\!], \ldots, [\![V_{r-1}]\!])^*f; [\![K]\!]$

Figure 10: Categorical semantics

f (equivalence class of terms)	$\overline{f}$ (an arbitrary member)
id	$\mathtt{x}_0$
f^*g	$\overline{g}[\overline{f}/\mathtt{x}_0]$
$g;h$ (g an $\mathcal{O}$-morphism)	$\overline{g} \bullet \overline{h}$
id	$\mathtt{nil}$
$h;k$ (h a $\mathcal{D}$-morphism)	$\overline{h} + \!\!+ \overline{k}$
π	$\pi\mathtt{x}_0$
π'	$\pi'\mathtt{x}_0$
(f,g)	$(\overline{f},\overline{g})$
force	$\mathtt{force}\ \mathtt{x}_0$
$\mathsf{q}^U f$	$\mathtt{thunk}\ \overline{f}$
return	$\mathtt{return}\ \mathtt{x}_0$
$\mathsf{q}^F f$	$[\cdot]\ \mathtt{to}\ \mathtt{x}_1.\ (\overline{f}[(\mathtt{x}_0,\mathtt{x}_1)/\mathtt{x}_0]) :: \mathtt{nil}$
$\mathsf{in}_{\hat{\imath}}$	$(\hat{\imath},\mathtt{x}_0)$
$\mathsf{q}^{\Sigma}\lambda i.g_i$	$\mathtt{pm}\ \mathtt{x}_0\ \mathtt{as}\ \{\ldots,(\mathtt{x}_1,(i,\mathtt{x}_2)).\overline{f_i}[(\mathtt{x}_1,\mathtt{x}_2)/\mathtt{x}_0],\ldots\}$
$\pi_{\hat{\imath}}$	$\hat{\imath} :: \mathtt{nil}$
$\mathsf{q}^{\Pi}\lambda i.g_i$ (g_i an $\mathcal{O}$-morphism)	$\lambda\{\ldots,i.\overline{g_i},\ldots\}$
$\mathsf{q}^{\Pi}\lambda i.h_i$ (h_i a $\mathcal{D}$-morphism)	$\{\ldots,\overline{h_i}\ \mathtt{where}\ i :: \mathtt{nil},\ldots\}\ \mathtt{is}\ \mathtt{nil}$
ev	$\mathtt{x}_0 :: \mathtt{nil}$
$\mathsf{q}^{\rightarrow}g$ (g an $\mathcal{O}$-morphism)	$\lambda\mathtt{x}_1.\overline{g}[(\mathtt{x}_0,\mathtt{x}_1)/\mathtt{x}_0]$
$\mathsf{q}^{\rightarrow}h$ (h a $\mathcal{D}$-morphism)	$\overline{h}[(\mathtt{x}_0,\mathtt{x}_1)/\mathtt{x}_0]\ \mathtt{where}\ \mathtt{x}_1 :: \mathtt{nil}\ \mathtt{is}\ \mathtt{nil}$

Figure 11: Classifying Model Of A Theory

We begin our account by defining an *on-the-nose morphism* from CBPV adjunction $(\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$ to CBPV adjunction $(\mathcal{C}', \mathcal{D}', \mathcal{O}', \ldots)$ in the obvious way. We then define **Adj** to be the category of CBPV adjunctions 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, 6 and 8, containing all the equations of Fig. 7, and closed under substitution, dismantling and concatenation.

8.3. Remark. An alternative formulation is followed in [Jef99, Lev96], 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. 7, 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. 7.3.) 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* G 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 $GA_0, \ldots, GA_{m-1} \vdash^{\mathsf{v}} W : GB$, and similarly for computations and stacks, preserving all the term constructors.

We can prove by induction that G 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.

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

Proof. Let $\mathcal{A} = (\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$ be a CBPV adjunction. 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

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

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 $\mathcal{A}$, using the definitions in Fig. 10. We then 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. 5.2 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 CBPV adjunction. The objects are just the types of τ, and the operations on objects given by the type structure. The homsets are defined by

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

All the categorical operations are defined in Fig. 11. Proving the equations of a CBPV adjunction 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

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

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

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

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. ∎

9. Call-By-Value is Kleisli, Call-By-Name is co-Kleisli

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

Kleisli

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

co-Kleisli

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 [LPT03] 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.

For the cpo model, the Kleisli category (over 1) provides the partial maps model for CBV [Plo85], while the co-Kleisli category (over 1) is pointed cpos and continuous maps—the standard model for CBN. Similarly from the game model (with the appropriate constraints of bracketing, visibility and innocence), we recover the standard CBV and CBN models [AM98, HY97, HO00, Nic96].

Notice that the duality, in a continuation model, between the Kleisli and co-Kleisli categories [Sel01, SR98] is a consequence of the duality between $\mathcal{C}$ and $\mathcal{D}_1$.

10. Stacks For Non-Algebraic Effects

Certain effects such as exception-handling are called *non-algebraic* in [PP01]. They present a problem for this work because they lead to additional stack terms that violate the laws we have imposed. As an example, for exception-handling we want the transition

$$\begin{array}{lllllr}
 & \Gamma \mid & \mathtt{try}\ M\ \mathtt{catch}\ e.N & FA & K & \underline{C} \\
\rightsquigarrow & \Gamma \mid & M & FA & \mathtt{try}\ [\cdot]\ \mathtt{catch}\ e.N :: K & \underline{C}
\end{array}$$

But consider the equation

$$K \mathbin{+\!\!+} L = [\cdot]\ \mathtt{to\ x.}\ ((\mathtt{return\ x}) \bullet {}^{\mathtt{x}}K) :: L$$

from Fig. 7. Put `try` $[\cdot]$ `catch` e.`return` $5 :: $ `nil` for K and `nil` for L. The two sides are clearly not equivalent: the LHS dismantled onto `raise` e returns 5, whereas the RHS dismantled onto `raise` e raises e. In terms of the algebra model for the $-+E$ monad on **Set**, a stack such as this K ought to denote a non-homomorphism.

This is roughly similar to the exception/continuation example in [Lai02]. More work is certainly required to find an appropriate equational theory and categorical structure for non-algebraic effects.

11. Further Directions

Most important is describing the universal properties in terms of naturality rather than elements, by means of a Yoneda Lemma. The trickiest in this regard is the distributive coproduct. See [Lev04] for further development of this.

In this paper we have presented the examples of CBPV adjunctions without any motivation, but of course they need to be explained operationally, especially as regards the stack morphisms; again, more information can be found in [Lev04].

The notion of stack judgement is in no way special to CBPV. It can be adapted to CBV; instead of CBPV adjunction we obtain "strong κ-category", presented in [LPT03] although not related there to the CK-machine. For CBN, the advantage is even greater. It appears that no categorical semantics for CBN has ever been given in the literature, because the weak coproducts denoted by sum types have a complicated structure. But stacks (provided only algebraic effects are allowed) resolve this problem: a family of CBN terms $\Gamma, \mathtt{x} : A_i \vdash M : B$, indexed by $i \in I$, corresponds to a stack $\Gamma | \sum_{i \in I} A_i \vdash^{\mathsf{k}} K : B$.

There are apparent connections between the models here and *polarized* models such as the game models in [CS, Lau02]. These need to be explored.

References

[AHM98] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proc., 13th Ann. IEEE Symp. on Logic in Comp. Sci.*, 1998.

[AM98] S. Abramsky and G. McCusker. Call-by-value games. In M. Nielsen and W. Thomas, editors, *Computer Science Logic: 11th International Workshop Proceedings*, LNCS. Springer, 1998.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proc., ACM Sigplan International Conference on Functional Programming (ICFP-00)*, volume 35.9 of *ACM Sigplan Notices*, pages 233–243, N.Y., September 18–21 2000. ACM Press.

[CLW93] A. Carboni, S. Lack, and R. F. C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84:145–158, 1993.

[CM93] P. Cenciarelli and E. Moggi. A syntactic approach to modularity in denotational semantics. In *CTCS 1993*, 1993.

[Coc93] J. R. B. Cockett. Introduction to distributive categories. *Mathematical Structures in Computer Science*, 3(3):277–307, September 1993.

[CS] R. Cockett and R. Seely. Polarized category theory, modules and game semantics. Preprint.

[Dan92] O. Danvy. Back to direct style. In *Proc. European Symposium on Programming*, volume 582 of *LNCS*, 1992.

[FF86] M. Felleisen and D. Friedman. Control operators, the SECD-machine, and the λ-calculus. In M. Wirsing, editor, *Formal Description of Prog. Concepts*. North-Holland, 1986.

[Fil96] A. Filinski. *Controlling Effects*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1996.

[FPD99] M. Fiore, 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)*, pages 193–202, Trento, Italy, July 1999. IEEE Computer Society Press.

[Har99] R. Harmer. *Games and full abstraction for nondeterministic languages*. PhD thesis, Univ. of London, 1999.

[HO00] M. Hyland and L. Ong. On full abstraction for PCF: I, II, and III. *Inf. and Comp.*, 163(2), 2000.

[HY97] K. Honda and N. Yoshida. Game theoretic analysis of call-by-value computation. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *LNCS*, pages 225–236, Bologna, Italy, 1997. Springer.

[Jef99] A. Jeffrey. A fully abstract semantics for a higher-order functional language with nondeterministic computation. *Theoretical Comp. Sci.*, 228, 1999.

[Kri85] J.-L. Krivine. Un interpréteur de λ-calcul. Unpublished, 1985.

[Lai97] J. Laird. Full abstraction for functional languages with control. In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 58–67, Warsaw, Poland, 1997. IEEE Computer Society Press.

[Lai98] J. Laird. *A Semantic Analysis of Control*. PhD thesis, University of Edinburgh, 1998.

[Lai02] James Laird. Exceptions, continuations and macro-expressiveness. *Lecture Notes in Computer Science*, 2305, 2002.

[Lau02] Olivier Laurent. Polarized games. In *Logic in Computer Science*, pages 265–274, Los Alamitos, CA, USA, July 22–25 2002. IEEE Computer Society.

[Law63] F. W. Lawvere. *Functorial Semantics of Algebraic Theories.* PhD thesis, Columbia University, 1963. Reprinted as Reprints in Theory and Applications of Categories, No. 5 (2004), pages 1–121.

[Lev] P. B. Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. submitted.

[Lev96] P. B. Levy. λ-calculus and cartesian closed categories. Essay for Part III of the Mathematical Tripos, Cambridge University, manuscript, 1996.

[Lev99] P. B. Levy. Call-by-push-value: a subsuming paradigm (extended abstract). In J.-Y Girard, editor, *Typed Lambda-Calculi and Applications*, volume 1581 of *LNCS*, 1999.

[Lev01] P. B. Levy. *Call-by-push-value.* PhD thesis, Queen Mary, University of London, 2001.

[Lev02] P. B. Levy. Possible world semantics for general storage in call-by-value. In J. Bradfield, editor, *Proc., 16th Annual Conference in Computer Science Logic, Edinburgh, 2002*, volume 2471 of *LNCS*, pages 232–246. Springer, 2002.

[Lev03] P. B. Levy. Adjunction models for call-by-push-value with stacks. In R. Blute and P. Selinger, editors, *Proc., 9th Conference on Category Theory and Computer Science, Ottawa, 2002*, volume 69 of *Electronic Notes in Theoretical Computer Science*, 2003.

[Lev04] P. B. Levy. *Call-By-Push-Value.* Semantic Structures in Computation. Kluwer, 2004.

[LPT03] P. B. Levy, A. J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Information and Computation*, 185:182–210, 2003.

[LRS93] Y. Lafont, B. Reus, and Th. Streicher. Continuation semantics or expressing implication by negation. Technical Report 9321, Ludwig-Maximilians-Universität, München, 1993.

[LS86] J. Lambek and P. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge University Press, Cambridge, 1986.

[Mog91] E. Moggi. Notions of computation and monads. *Inf. and Comp.*, 93, 1991.

[Mog90] E Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Dept. of Computer Science, Edinburgh Univ., 90.

[Nic96] H. Nickau. *Hereditarily Sequential Functionals: A Game-Theoretic Approach to Sequentiality*. Shaker-Verlag, 1996. Diss., Universität Gesamthochschule Siegen.

[Ole82] F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages*. Ph. D. dissertation, Syracuse University, August 1982.

[Plo85] G. D. Plotkin. Lectures on predomains and partial functions. Course notes, Center for the Study of Language and Information, Stanford, 1985.

[PP01] Gordon Plotkin and John Power. Adequacy for algebraic effects. *Lecture Notes in Computer Science*, 2030, 2001.

[PS98] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 227–273. Cambridge University Press, 1998.

[Sel01] P. Selinger. Control categories and duality: On the categorical semantics of the $\lambda\mu$-calculus. *Mathematical Structures in Computer Science*, 11(2), 2001.

[SF93] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4), 1993.

[SR98] Th. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6), 1998.

[Thi97] H. Thielecke. Continuation semantics and self-adjointness. In *Proceedings MFPS XIII*, Electronic Notes in Theoretical Computer Science. Elsevier, 1997.

School of Computer Science, University of Birmingham
Birmingham B15 2TT, United Kingdom
Email: `pbl@cs.bham.ac.uk`

This article may be accessed via WWW at `http://www.tac.mta.ca/tac/` or by anonymous ftp at `ftp://ftp.tac.mta.ca/pub/tac/html/volumes/14/5/14-05.{dvi,ps}`

THEORY AND APPLICATIONS OF CATEGORIES (ISSN 1201-561X) will disseminate articles that significantly advance the study of categorical algebra or methods, or that make significant new contributions to mathematical science using categorical methods. The scope of the journal includes: all areas of pure category theory, including higher dimensional categories; applications of category theory to algebra, geometry and topology and other areas of mathematics; applications of category theory to computer science, physics and other mathematical sciences; contributions to scientific knowledge that make use of categorical methods.

Articles appearing in the journal have been carefully and critically refereed under the responsibility of members of the Editorial Board. Only papers judged to be both significant and excellent are accepted for publication.

The method of distribution of the journal is via the Internet tools WWW/ftp. The journal is archived electronically and in printed paper format.

Subscription information. Individual subscribers receive (by e-mail) abstracts of articles as they are published. Full text of published articles is available in .dvi, Postscript and PDF. Details will be e-mailed to new subscribers. To subscribe, send e-mail to tac@mta.ca including a full name and postal address. For institutional subscription, send enquiries to the Managing Editor.

Information for authors. The typesetting language of the journal is TeX, and LaTeX2e is the preferred flavour. TeX source of articles for publication should be submitted by e-mail directly to an appropriate Editor. They are listed below. Please obtain detailed information on submission format and style files from the journal's WWW server at http://www.tac.mta.ca/tac/. You may also write to tac@mta.ca to receive details by e-mail.