Call-By-Push-Value: Decomposing Call-By-Value And Call-By-Name

Paul Blain Levy (pbl@cs.bham.ac.uk)
University of Birmingham

Abstract. We present the call-by-push-value (CBPV) language paradigm, which decomposes the typed call-by-value (CBV) and typed call-by-name (CBN) paradigms into fine-grain primitives. On the operational side, we give big-step semantics and a stack machine for CBPV, which leads to a straightforward push/pop reading of CBPV programs. On the denotational side, we model CBPV using cpos and, more generally, using algebras for a strong monad. For storage, we present an O'Hearn-style "behaviour semantics" that does not use a monad.

We present the translations from CBN and CBV to CBPV. All these translations straightforwardly preserve denotational semantics. We also study their operational properties: simulation and full abstraction.

We give an equational theory for CBPV, and show it equivalent to a categorical semantics using monads and algebras We use this theory to formally compare CBPV to Filinski's variant of the monadic metalanguage, as well as to Marz' language SFPL, both of which have essentially the same type structure as CBPV. We also discuss less formally the differences between the CBPV and monadic frameworks.

1. Introduction

1.1. Aims of Paper

Let us consider typed call-by-value (CBV) and typed call-by-name (CBN), and observe convergence at ground type only. (This restriction does not matter in CBV, but in CBN, it makes the η-law for functions into an observational equivalence.) Suppose we seek to combine these into a single "subsuming" language such that

- the subsuming language, like CBV and CBN, is equipped with operational semantics, cpo semantics, monad semantics, storage semantics in the manner of (O'Hearn, 1993) and continuation semantics in the manner of (Streicher and Reus, 1998)
- these semantics are at least as simple as the corresponding semantics for CBV and CBN
- the translations preserve all these semantics.

We could add "etc." to the list of semantics, but for the sake of precision we will stop there.

The reason this is a desirable objective is that it is *plausible* that the situation found for all the semantics listed will also be true for all other CBV and CBN semantics we might wish to study. (This cannot be made into a precise statement, because of the simplicity requirement.) If so, then a researcher studying a new kind of semantics need only develop it for the subsuming language, because the CBV and CBN semantics can be derived from the subsuming semantics.

In this paper, we introduce a calculus, *call-by-push-value* (CBPV), which is a solution to this problem. It was obtained by analyzing the above semantics to find comon underlying primitives. But this paper does not follow that route; instead, the only knowledge presupposed is big-step and cpo semantics for CBV and CBN, and global store and monad semantics for CBV.

1.2. Related Work

CBPV is closely related to Filinski's Effect-PCF (Filinski, 1996), a form of the monadic metalanguage (Moggi, 1991). However it differs from Effect-PCF in 2 respects.

1. CBPV's computation types denote algebras, not merely carriers of algebras. As we explain in Sect. 2.1, this is essential in order to treat CBN compositionally.
2. CBPV retains the distinction between a computation and its thunk, familiar to CBV programmers but erased in monadic metalanguages. Sect. 2.2 explains this point in the more familiar CBV setting, before we come to CBPV.

Besides Effect-PCF, and somewhat similar pointed/unpointed calculi such as (Howard, 1996), there has been much work bringing CBV and CBN into a common framework. However, it is usually with regard to a narrower range of semantics than we are considering.

- Translations into intuitionistic linear logic (Benton and Wadler, 1996) preserve cpo semantics, but not the others.
- Translations into SFPL (Marz, 2000) preserve cpo semantics and operational semantics, but not the others.
- Translations into continuation languages (Plotkin, 1976), or their polarized counterpart LLP (Laurent, 1999), preserve continuation semantics, including unbracketed game semantics (Laird, 1998), and (certain) operational semantics, but not the others. Likewise the related work of (Selinger, 2001).

We therefore emphasize what, by contrast, is extraordinary about CBPV: the translations into it preserve such a wide range of semantics. Indeed there are many more, including game semantics, possible world semantics, non-monad models of nondeterminism, etc., that we do not treat in this paper, and the reader is referred to (Levy, 2004) for more information.

1.3. Structure of Paper

Before looking at CBPV, we develop some themes in monad semantics of CBN (Sect. 2.1) and CBV (Sect. 2.2). We then introduce CBPV, with its monad/algebra semantics, and big-step semantics. We also present a stack machine, which explains why functions are computations in CBPV: they pop their argument from the stack.

A novel part of the paper is Sect. 6.2, which presents a *behaviour semantics* for storage—actually the simpler CBPV version of the CBN semantics in (O'Hearn, 1993). Because it is not a monad semantics, this model illustrates the difference between CBPV and the monadic framework. Furthermore, we see that its soundness is trivial, by contrast with that of the monad/algebra model, whose proof we defer.

We then give the translations from CBV and CBN into CBPV, and prove preservation of denotational semantics. This is a trivial result, but it is the most important one, in the light of our original problem. We also show preservation of operational semantics—not exactly, but up to some minor "administrative reductions". And we state (without proof) full abstraction theorems for these translations.

Next, we move to the more technical part of the paper: relating CBPV to the well-established monadic framework. To do this, we need an equational theory for CBPV, and formulating it requires us to add certain *complex values* to the syntax, though they can always be eliminated from a computation. We give a monad/algebra categorical semantics and prove that every CBPV model is equivalent to a monad model (even though it may be unnatural to present it in this way). This enables us, finally, to prove the soundness of the algebra model for storage, deducing it from that of the behaviour semantics.

Finally, having dealt with complex values and the equational theory, it is straightforward to relate CBPV to Filinski's Effect-PCF and Marz' SFPL.

2. Monads

2.1. Algebra Semantics For Call-By-Name

Moggi's seminal paper (Moggi, 1991) provided translations from CBV and CBN into his "monadic metalanguage", and hence semantics for CBV and CBN in any bicartesian closed category $\mathcal{C}$ equipped with a strong monad T. The CBN translation given there is shown in (Hatcliff, 1994) to be the composite

$$\text{CBN} \longrightarrow \text{CBV} \longrightarrow \text{monadic metalanguage}$$

Here the first factor is the *thinking transform* of (Hatcliff and Danvy, 1997), which breaks the η-law for functions, and the similar law $\lambda \mathtt{x}_A.\mathtt{diverge}_B = \mathtt{diverge}_{A\to B}$.

A different semantics of CBN in $(\mathcal{C}, T)$, which we shall call *carrier semantics* for reasons to be seen shortly, is given in (Benton et al., 2000; Filinski, 1996). In it, we have

$$\begin{aligned} [\![\mathtt{bool}]\!] &= T(1+1) \\ [\![A \to B]\!] &= [\![A]\!] \to [\![B]\!] \\ [\![A + B]\!] &= T([\![A]\!] + [\![B]\!]) \end{aligned}$$

and a term $A_0, \ldots, A_{n-1} \vdash M : B$ denotes a function from $[\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$ to $[\![B]\!]$.

This semantics does validate the η-law for functions, but it is not compositional. For example, the interpretation of `if` must be given by induction over types: it is trivial at ground type or sum type, and at function type it is given by

$$\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N' \;=\; \lambda \mathtt{x}.\mathtt{if}\ M\ \mathtt{then}\ (\mathtt{x}`N)\ \mathtt{else}\ (\mathtt{x}`N') \quad (1)$$

Furthermore, in order to prove the correctness of the semantics, it is necessary to prove (1) valid as an observational equivalence—a job we really want done for us by a denotational model.

The solution to this non-compositionality problem is that, in monad semantics, a CBN type should denote not an object of $\mathcal{C}$, but rather a T-*algebra*. We recall that this is defined to be a pair (X, θ), where $X \in \mathsf{ob}\,\mathcal{C}$ and $TX \xrightarrow{\theta} X$ is a morphism such that

$$\begin{tikzcd} X \arrow[r, "\eta X"] \arrow[dr, "\mathsf{id}"'] & TX \arrow[d, "\theta"] & T^2X \arrow[l, "\mu X"'] \arrow[d, "T\theta"] \\ & X & TX \arrow[l, "\theta"] \end{tikzcd} \quad (2)$$

commutes. We call X the *carrier* and θ the *structure* of the algebra. Here are some examples:

- An algebra for the lifting monad on **Cpo** has a *pointed cpo* or *cppo* (cpo with a least element), as a carrier, and each pointed cpo has a unique structure map. Thus this monad is unusual in that an algebra is determined by its carrier.
- An algebra for the printing monad $\mathcal{A}^* \times -$ on **Set** can be described as an $\mathcal{A}$-*set*, a set X together with a binary operation $*$ from $\mathcal{A} \times X$ to X.

Given a strong monad on cartesian $\mathcal{C}$, we can build algebras for it in the following ways:

- every object A gives a *free* algebra $(TA, \mu A)$
- every family of algebras $\{(X_i, \theta_i)\}_{i \in I}$ has a *product algebra*, with carrier $\prod_{i \in I} X_i$, provided this product exists in $\mathcal{C}$
- from every object A to every algebra (X, θ) there is an *exponential algebra*, with carrier $A \to X$, provided this exponential exists in $\mathcal{C}$.

Suppose we write $F^T A$ for the free T-algebra on A, and $U^T \underline{B}$ for the carrier of a T-algebra $\underline{B}$. Then (assuming $\mathcal{C}$ to be bicartesian closed) the semantics of CBN types is given by

$$\begin{aligned} [\![\texttt{bool}]\!] &= F^T(1+1) \\ [\![A \to B]\!] &= U^T [\![A]\!] \to [\![B]\!] \\ [\![A + B]\!] &= F^T(U^T [\![A]\!] + U^T [\![B]\!]) \end{aligned}$$

and a term $A_0, \ldots, A_{n-1} \vdash M : B$ denotes a function from $U^T [\![A_0]\!] \times \cdots \times U^T [\![A_{n-1}]\!]$ to the carrier of $[\![B]\!]$.

Clearly, every type's carrier denotation is the carrier of its algebra denotation (hence the name "carrier semantics"), and every term has the same denotation in the 2 semantics. But in algebra semantics, the interpretation of conditionals is compositional.

Similarly, consider a CBN language containing a `print`c instruction (for $c \in \mathcal{A}$) that can be prefixed to a term of any type. The language is modelled using the printing monad $\mathcal{A}^* \times -$ on **Set**. In carrier semantics, `print` would be interpreted by induction on types. But in algebra semantics, a CBN type denotes an $\mathcal{A}$-set $(X, *)$, and we define

$$[\![\texttt{print}\ c.\ M]\!]\rho = c * [\![M]\!]\rho$$

The following is sufficient to ensure that all exponentials and finite products of algebras exist, so that algebra semantics can be constructed.

Definition 1. An *algebra-building structure* consists of a strong monad (T, η, μ, t) on a distributive category $\mathcal{C}$, with an exponential from every $\mathcal{C}$-object to every carrier of a T-algebra.

2.2. Call-By-Value and Thunks

We recapitulate and critique the analysis of CBV semantics that leads to the monadic metalanguage.

Consider a CBV language with booleans and multi-ary functions, together with some computational effects—let us say global store, and write S for the set of stores. The types of this language are

$$A ::= \ \mathtt{bool} \mid A_0, \ldots, A_{n-1} \to A$$

The 0-ary function type $\to A$ is well known to CBV programmers, being a type of *thunks* that can be forced whenever convenient. Following (Hatcliff and Danvy, 1997; Moggi, 1988), we might call this type TA, and write `thunk` M and `force` N for $\lambda[\,].M$ and $N[\,]$ respectively.

Before monads became popular, the denotational semantics of this language would have been formulated as follows. First we give the semantics of types, so that a type denotes a set. Then we give 2 semantic functions on terms:

– a function $[\![-]\!]^{\mathsf{ret}}$ taking each term $\Gamma \vdash M : B$ to a function $S \times [\![\Gamma]\!] \xrightarrow{[\![M]\!]^{\mathsf{ret}}} S \times [\![B]\!]$

– a function $[\![-]\!]^{\mathsf{val}}$ taking each value $\Gamma \vdash V : B$ to a function $[\![\Gamma]\!] \xrightarrow{[\![A]\!]^{\mathsf{val}}} [\![B]\!]$.

Next we prove $[\![V]\!](s, \rho) = (s, [\![V]\!]\rho)$, for each value $\Gamma \vdash V : B$. We proceed to prove 2 substitution lemmas, for substitution of values into terms and into values. Finally, we prove soundness and adequacy, which completes the story.

The line of thought that leads from this account to the monadic metalanguage proceeds in two steps.

The first step is as follows. Since a value has 2 denotations—as a term, and as a value—it makes sense to introduce an explicit judgement $\Gamma \vdash^{\mathsf{v}} V : A$ for values, and explicitly coerce values into effectful terms.

Doing this gives something like the *fine-grain CBV* language[1] shown in Fig. 1.

Types

The same as the original CBV language

Judgements

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

Terms

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

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

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

The rules for multi-ary functions are similar.

$$\frac{}{\Gamma \vdash^{\mathsf{v}} \mathtt{true} : \mathtt{bool}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{bool} \quad \Gamma \vdash M : B \quad \Gamma \vdash M' : B}{\Gamma \vdash \mathtt{if}\ V\ \mathtt{then}\ M\ \mathtt{else}\ M' : B}$$

Figure 1. Syntax of Fine-Grain CBV, For Boolean and Multi-ary Function Types

The second step is to observe that

$$\begin{aligned} [\![\to B]\!] &= S \to (S \times [\![B]\!]) \\ [\![\lambda[\,].M]\!]\rho &= \lambda s.([\![M]\!]s\rho) \\ [\![V[\,]]\!]s\rho &= ([\![V]\!]\rho)s \end{aligned}$$

so terms $\Gamma \vdash M : B$ correspond to values $\Gamma \vdash^{\mathsf{v}} V :\to B$, via these thunking and forcing operations. Therefore—it is argued—the $\vdash$ judgement is redundant, and we might as well abolish it, leaving only values. That gives the monadic metalanguage.

But, whereas the first step greatly simplifies the semantics of CBV, the second step is problematic. Firstly, because it erases the conceptually significant difference between a CBV term and its thunk. Secondly,

[1] The language MIL-lite in (Benton and Kennedy, 2000), which distinguishes between different effects, is somewhat similar. However, fine-grain CBV differs from the original CBV language only in its judgements and terms; the types are unchanged.

because this difference, though invisible in monad semantics, is apparent in many others. These other semantics, although they *can* be squashed into the monadic straitjacket, become less simple and less intuitive as a consequence.

The most glaring example is the possible world semantics of (Levy, 2002), where the semantic equations for the monadic metalanguage are much more complicated than for fine-grain CBV, because they must repeatedly force and thunk. Other examples are continuation and game models; these describe the interaction or jumping between different parts of a program, and forcing corresponds to a jump. It is not possible to treat these examples in this paper, and a full treatment can be found in (Levy, 2004).

For these reasons, we will maintain the distinction between a term (computation) and its thunk.

Remark 1. If we had included in Fig. 1 the typing rule

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \texttt{bool} \quad \Gamma \vdash^{\mathsf{v}} W : B \quad \Gamma \vdash^{\mathsf{v}} W' : B}{\Gamma \vdash^{\mathsf{v}} \texttt{if}\ V\ \texttt{then}\ W\ \texttt{else}\ W' : B}$$

then values themselves would need to be evaluated, and the operational semantics would become more complicated. However, in the equational theory for fine-grain CBV, it can be shown that such “complex values” can be eliminated from any effectful term. We shall see (Sect. 8) a similar situation for CBPV.

3. CBPV Syntax and Monad/Algebra Semantics

CBPV has two disjoint classes of terms: values and computations. Below, we shall see this difference in operational terms: a value *is*, whereas a computation *does*. As explained in Sect. 2.2, we take care to distinguish a computation M from its *thunk*, the latter being a value that can be *forced* at any time.

CBPV likewise has two disjoint classes of type: a value has a value type, while a computation has a computation type. The types are given by

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

where I can be any finite[2] set.

[2] For certain purposes, including game semantics, Böhm trees and possible worlds, it is convenient to consider infinitary forms of CBPV, CBV and CBN. In the

It is obvious how to interpret these types in an algebra-building structure: a value type denotes an object of $\mathcal{C}$ whereas a computation type denotes a T-algebra. Thus FA denotes the free T-algebra on $[\![A]\!]$, whilst $U\underline{B}$ denotes the carrier of $[\![\underline{B}]\!]$. The type $A \to \underline{B}$ denotes the exponential algebra from $[\![A]\!]$ to $[\![\underline{B}]\!]$.

In particular, using the lifting monad on **Cpo**, we interpret a value type by a cpo and a computation type by a cppo. Here FA denotes the lift of $[\![A]\!]$, whilst $U\underline{B}$ has the same denotation as $\underline{B}$, so U is invisible. (A unary construct c is said to be *invisible* in a given denotational semantics when $[\![c(Q)]\!] = [\![Q]\!]$.)

As in CBV, 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. 2. We explain some of the less familiar constructs. The keyword `pm` stands for "pattern-match". We write ' for application in reverse order; the advantage of this is explained in Sect. 5.2. 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 $\to$. We use the keyword `to` corresponding to the Haskell symbol `>>=`. Thus M `to x.` N is the sequenced computation that first executes M, and when this produces a value `x` proceeds to execute N. We reserve `let` for plain binding.

In an algebra-building structure $(\mathcal{C}, T)$, a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$, whilst a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to the carrier of $[\![\underline{B}]\!]$. In particular:

– if $\Gamma \vdash^{\mathsf{v}} V : A$, then `return` V denotes the composite

$$[\![\Gamma]\!] \xrightarrow{[\![V]\!]} [\![A]\!] \xrightarrow{\eta[\![A]\!]} T[\![A]\!]$$

infinitary setting, I can be any countable set (but only finite products of value types are allowed). In this paper, we treat only finitary languages, but use the indexed notation with a view to this infinitary extension.

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

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

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

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

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

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

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

Figure 2. Terms of CBPV

- If $\Gamma \vdash^{\mathsf{c}} M : FA$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}$ and $\underline{B}$ denotes the algebra (Y, ϕ) then M `to x.` N denotes the composite

$$\begin{tikzcd}
[\![\Gamma]\!] \arrow[r, "(\mathsf{id}, [\![M]\!])"] & [\![\Gamma]\!] \times T[\![A]\!] \arrow[r, "t([\![\Gamma]\!], [\![A]\!])"] & T([\![\Gamma]\!] \times [\![A]\!]) \arrow[r, "T[\![N]\!]"] & TY \arrow[d, "\phi"] \\
& & & Y
\end{tikzcd}$$

- `thunk` and `force` are invisible.

In Sect. 6.2, we shall see a model where `thunk` and `force` are visible.

In Fig. 3 we show how to add constructs for divergence/recursion, printing elements of a set, and storing elements of a set in a global cell. Although there are many other effects we can treat, this limited range suffices to illustrate our main points about CBPV.

It is convenient to treat commands for printing etc. as prefixes, rather than as primitive terms.

Divergence

$$\frac{}{\Gamma \vdash^{\mathsf{c}} \texttt{diverge} : \underline{B}} \qquad \frac{\Gamma, \mathtt{x} : U\underline{B} \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mu\mathtt{x}.M : \underline{B}}$$

Printing elements of a countable set $\mathcal{A}$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \texttt{print}\ c.\ M : \underline{B}}$$

Storing elements of a finite set S in a cell
(We could allow denumerable S, making the syntax infinitary.)

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \texttt{cell} := s.\ M : \underline{B}} \qquad \frac{\cdots\ \Gamma \vdash^{\mathsf{c}} M_s : \underline{B}\ \cdots_{\ s \in S}}{\Gamma \vdash^{\mathsf{c}} \texttt{read-cell-as}\ \{\ldots, s.M_s, \ldots\} : \underline{B}}$$

Figure 3. Adding Divergence, Printing, Storage

The denotational semantics of divergence in the cppo model is

$$[\![\texttt{diverge}]\!]\rho \ = \ \bot$$

with $\mu\mathtt{x}.M$ interpreted as a least prefixpoint. The denotational semantics of printing in the $\mathcal{A}$-set model is

$$[\![\texttt{print}\ c.\ M]\!]\rho \ = \ c * [\![M]\!]$$

We discuss denotational semantics of storage in Sect. 6.1.

4. Big-Step Semantics

As in CBN we begin the big-step semantics by defining a special class of closed computations where evaluation stops, which we call *terminal computations*. (We cannot, of course, call them "values".) They are given by

$$T ::= \ \texttt{return}\ V \ \mid\ \lambda\{\ldots, i.M_i, \ldots\} \ \mid\ \lambda\mathtt{x}.M$$

The big-step semantics are expressed using the judgement $M \Downarrow T$, where M is a closed computation and T a terminal computation of the same type. The rules are presented in Fig. 4.

Each big-step rule has the form

$$\frac{M_0 \Downarrow T_0 \quad \cdots \quad M_{r-1} \Downarrow T_{r-1}}{M \Downarrow T} \tag{3}$$

for some $r \geqslant 0$. Here they are:

$$\frac{M[V/\mathtt{x}] \Downarrow T}{\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M \Downarrow T}$$

$$\frac{}{\mathtt{return}\ V \Downarrow \mathtt{return}\ V} \qquad \frac{M \Downarrow \mathtt{return}\ V \quad N[V/\mathtt{x}] \Downarrow T}{M\ \mathtt{to}\ \mathtt{x}.\ N \Downarrow T}$$

$$\frac{M \Downarrow T}{\mathtt{force}\ \mathtt{thunk}\ M \Downarrow T}$$

$$\frac{M_{\hat{\imath}}[V/\mathtt{x}] \Downarrow T}{\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\} \Downarrow T}$$

$$\frac{M[V/\mathtt{x}, V'/\mathtt{y}] \Downarrow T}{\mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M \Downarrow T}$$

$$\frac{}{\lambda\{\ldots, i.M_i, \ldots\} \Downarrow \lambda\{\ldots, i.M_i, \ldots\}} \qquad \frac{M \Downarrow \lambda\{\ldots, i.N_i, \ldots\} \quad N_{\hat{\imath}} \Downarrow T}{\hat{\imath}`M \Downarrow T}$$

$$\frac{}{\lambda\mathtt{x}.M \Downarrow \lambda\mathtt{x}.M} \qquad \frac{M \Downarrow \lambda\mathtt{x}.N \quad N[V/\mathtt{x}] \Downarrow T}{V`M \Downarrow T}$$

Figure 4. Big-Step Semantics for CBPV

Proposition 1. (Termination and Determinism) **no effects** For each M there is a unique T such that $M \Downarrow T$.

divergence For each M there exists at most one T such that $M \Downarrow T$.

printing For each M there is a unique m, T such that $M \Downarrow m, T$.

storage For each s, M there is a unique s', T such that $s, M \Downarrow s', T$.

The proof is standard, and in the Appendix.

Divergence We add the rules

$$\frac{\texttt{diverge} \Downarrow T}{\texttt{diverge} \Downarrow T} \qquad \frac{M[\texttt{thunk}\ \mu\texttt{x}.M/\texttt{x}] \Downarrow T}{\mu\texttt{x}.M \Downarrow T}$$

and we say that $M \Uparrow$ when there does not exist T such that $M \Downarrow T$. (This exploits determinism, of course.)

Printing The big-step judgement takes the form $M \Downarrow m, T$ where $m \in \mathcal{A}^*$. We accordingly replace each rule (3) by

$$\frac{M_0 \Downarrow m_0, T_0 \quad \cdots \quad M_{r-1} \Downarrow m_{r-1}, T_{r-1}}{M \Downarrow m_0 * \ldots * m_{r-1}, T}$$

and we add the rule

$$\frac{M \Downarrow m, T}{\texttt{print}\ c.\ M \Downarrow c * m, T}$$

Storage The big-step judgement takes the form $s, M \Downarrow s', T$ where $s, s' \in S$. We accordingly replace each big-step rule of the form (3) by

$$\frac{s_0, M_0 \Downarrow s_1, T_0 \quad \cdots \quad s_{r-1}, M_{r-1} \Downarrow s_r, T_{r-1}}{s_0, M \Downarrow s_r, T}$$

and we add the rules

$$\frac{s', M \Downarrow s'', T}{s, \texttt{cell} := s'.\ M \Downarrow s'', T} \qquad \frac{s', M_{s'} \Downarrow s'', T <}{s', \texttt{read-cell-as}\ \{\ldots, s.M_s, \ldots\} \Downarrow s'', T}$$

Figure 5. Big-Step Semantics For Divergence, Printing and Storage

Definition 2. Let $\Gamma \vdash^{\mathsf{c}} M, M' : \underline{B}$ be two computations. By *ground context* we mean a computation of type $F\sum_{i \in I} 1$ with some occurrences of a hole $\Gamma \vdash^{\mathsf{c}} [\cdot] : \underline{B}$.

no effects $M \simeq M'$ when $\mathcal{C}[M] \Downarrow \texttt{return}\ a$ iff $\mathcal{C}[M'] \Downarrow \texttt{return}\ a$ for every ground context $\mathcal{C}[\cdot]$

divergence $M \lesssim M'$ when $\mathcal{C}[M] \Downarrow \texttt{return}\ a$ implies $\mathcal{C}[M'] \Downarrow \texttt{return}\ a$ for every ground context $\mathcal{C}[\cdot]$

printing $M \simeq M'$ when $\mathcal{C}[M] \Downarrow m, \texttt{return}\ a$ iff $\mathcal{C}[M'] \Downarrow m, \texttt{return}\ a$ for every ground context $\mathcal{C}[\cdot]$

storage $M \simeq M'$ when $s, \mathcal{C}[M] \Downarrow s', \texttt{return}\ a$ iff $s, \mathcal{C}[M'] \Downarrow s', \texttt{return}\ a$ for every $s, s' \in S$ and every ground context $\mathcal{C}[\cdot]$.

Similarly for two values $\Gamma \vdash^{\mathsf{v}} V, V' : A$.

In Sect. 3 we have given denotational semantics for divergence (using cpos/cppos) and for printing (using $\mathcal{A}$-sets), and we have to prove them sound and adequate.

Proposition 2. (soundness/adequacy) Let M be a closed computation.

divergence If $M \Downarrow T$ then $[\![M]\!] = [\![T]\!]$. If M diverges then $[\![M]\!]^{\mathsf{ret}} = \bot$.

printing If $M \Downarrow m, T$ then $[\![M]\!] = m * [\![T]\!]$.

The proof is standard, and in the Appendix.

Corollary 1. Let M be a closed ground returner.

divergence $[\![M]\!] = \mathsf{lift}\ a$ iff $M \Downarrow \mathtt{return}\ a$. Similarly $[\![M]\!] = \bot$ iff M diverges.

printing $[\![M]\!] = (m, a)$ iff $M \Downarrow m, \mathtt{return}\ a$.

Hence terms with the same denotation are observationally equivalent. In the case of the divergence model, if $[\![M]\!] \leqslant [\![N]\!]$ then $M \lesssim N$.

5. CK-Machine

5.1. Introducing The CK-Machine

The CK-machine is a form of operational semantics that is more explicit than big-step semantics and has certain advantages over it; for example, it allows the easy formulation of control effects. It can be given for CBV, CBN and CBPV. It was introduced by (Felleisen and Friedman, 1986) in a CBV setting, and there are many similar formulations (Krivine, 1985; Pitts and Stark, 1998; Streicher and Reus, 1998). The CK-machine semantics for CBPV is given in Fig. 6.

At any point in time, the machine has configuration M, K when M is the computation we are evaluating and K is a stack of contexts. In this stack, we abbreviate the context $V`[\cdot]$ as $V$, and the context $\hat{\imath}`[\cdot]$ as $\hat{\imath}$.

To understand the CK-machine, just think about how we might implement the big-step rules using a stack. Suppose for example that we are evaluating M $\mathtt{to}\ \mathtt{x}.\ N$. The big-step semantics tells us that we must first evaluate M. So we put the context $[\cdot]$ $\mathtt{to}\ \mathtt{x}.\ N$ onto the

Initial Configuration

	M	$\texttt{nil}$

Transitions

	$\texttt{let } V \texttt{ be x. } M$	K
$\rightsquigarrow$	$M[V/\texttt{x}]$	K
	$M \texttt{ to x. } N$	K
$\rightsquigarrow$	M	$[\cdot] \texttt{ to x. } N :: K$
	$\texttt{return } V$	$[\cdot] \texttt{ to x. } N :: K$
$\rightsquigarrow$	$N[V/\texttt{x}]$	K
	$\texttt{force thunk } M$	K
$\rightsquigarrow$	M	K
	$\texttt{pm } (\hat{\imath}, V) \texttt{ as } \{\ldots, (i, \texttt{x}).M_i, \ldots\}$	K
$\rightsquigarrow$	$M_{\hat{\imath}}[V/\texttt{x}]$	K
	$\texttt{pm } (V, V') \texttt{ as } (\texttt{x}, \texttt{y}).M$	K
$\rightsquigarrow$	$M[V/\texttt{x}, V'/\texttt{y}]$	K
	$\hat{\imath}`M$	K
$\rightsquigarrow$	M	$\hat{\imath} :: K$
	$\lambda\{\ldots, i.M_i, \ldots\}$	$\hat{\imath} :: K$
$\rightsquigarrow$	$M_{\hat{\imath}}$	K
	$V`M$	K
$\rightsquigarrow$	M	$V :: K$
	$\lambda\texttt{x}.M$	$V :: K$
$\rightsquigarrow$	$M[V/\texttt{x}]$	K

Terminal Configurations

	$\texttt{return } V$	$\texttt{nil}$
	$\lambda\{\ldots, i.M_i, \ldots\}$	$\texttt{nil}$
	$\lambda\texttt{x}.M$	$\texttt{nil}$

Figure 6. CK-Machine For CBPV

stack, because at present we do not need it. Later, having evaluated M to `return` V, we can remove $[\cdot]$ `to x.` N from the stack and proceed to evaluate $N[V/\mathtt{x}]$, as the big-step semantics suggests.

As another example, suppose we are evaluating $V`M$. The big-step semantics tells us that we must first evaluate $M$. So we put the context $V`[\cdot]$ (abbreviated as V) onto the stack, because at present we do not need it. Later, having evaluated M to $\lambda\mathtt{x}.N$, we can remove this context from the stack and proceed to evaluate $N[V/\mathtt{x}]$, as the big-step semantics suggests.

To evaluate a closed computation M, we start with the configuration $M, \mathtt{nil}$ and follow the transitions in Fig. 6 until we reach a configuration $T, \mathtt{nil}$ for a terminal computation T.

The CK-machine agrees with the big-step semantics in the following sense:

Proposition 3. For any closed computation M, we have $M \Downarrow T$ iff $M, \mathtt{nil} \rightsquigarrow^* T, \mathtt{nil}$.

This is proved in (Levy, 2004) by standard techniques. For each of our effects, it is straightforward to adapt the CK-machine and obtain a variant of Prop. 3.

5.2. Pushing and Popping

The strangest feature of CBPV, for people familiar with CBV, is the fact that $\lambda\mathtt{x}.M$ is a computation. But the CK-machine gives a simple explanation of this feature. Looking at Fig. 6, it is apparent that $V`$ can be read as an instruction "push V" whilst $\lambda\mathtt{x}$ can be read as an instruction "pop $\mathtt{x}$". This is why we prefer an operand-first notation for application.

A fortunate consequence of the push/pop interpretation is that it makes CBPV programs easy to read. Here is an example program using printing. The program involves some complex values such as arithmetic and string expressions, which are easy to understand although they are not officially included within the CK-machine.

```
print "hello0".
let 3 be x.
let  thunk (
        print "hello1".
        λz.
        print "we just popped "z.
        produce x + z
     ) be y.
```

```
print "hello2".
( print "hello3".
  7‘
  print "we just pushed 7".
  force y
) to w.
print "w is bound to "w.
return w + 5
```

It is easy to see that the program outputs as follows

```
hello0
hello2
hello3
we just pushed 7
hello1
we just popped 7
w is bound to 10
```

and finally returns the value 15.

From this viewpoint, we can give the following operational summary of CBPV types.

- A value of type $U\underline{B}$ is a thunk of a computation of type $\underline{B}$.
- A value of type $\sum_{i \in I} A_i$ is a pair (i, V), where $i \in I$ and V is a value of type A_i.
- A value of type $A \times A'$ is a pair (V, V'), where V is a value of type A and V' is a value of type A'.
- A value of type 1 is the 0-tuple $()$.
- A computation of type FA returns a value of type A.
- A computation of type $\prod_{i \in I} \underline{B}_i$ pops a tag $i \in I$, and then behaves as a computation of type $\underline{B}_i$.
- A computation of type $A \to \underline{B}$ pops a value of type A and then behaves as a computation of type $\underline{B}$.

Notice how this description follows the principle “a value is, a computation does”.

6. Denotational Semantics For Storage

6.1. Monad/Algebra Semantics

So far we have seen denotational semantics for divergence and printing, and proved them sound and adequate, but what about storage? One seemingly reasonable way of building such a semantics is to use the $S \to (S \times -)$ monad on **Set** in the manner of Sect. 3, so that a computation type denotes an algebra for this monad. But we still have the task of proving some kind of soundness theorem, at the very least the following.

Proposition 4. If M is a closed computation of type FA, and $s, M \Downarrow s', \texttt{return}\ V$, then $[\![M]\!]s = (s', [\![V]\!])$.

This is not straightforward to prove. The easiest known method is to introduce another denotational model, prove the latter sound, and then prove the agreement of the two models (we describe this agreement at the end of Sect. 9). We now turn to this other denotational model, called *behaviour semantics*, and introduced for CBN in (O'Hearn, 1993). Not only is it easier to prove sound, but it is arguably more intuitive than the algebra semantics.

6.2. Behaviour Semantics

For values, behaviour semantics is no different from monad semantics:

- a value type A (and similarly a context Γ) denotes a set
- the connectives $\sum$ and $\times$ denote sum and product of sets
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a function from $[\![\Gamma]\!]$ to $[\![A]\!]$.

But a computation type $\underline{B}$ denotes not an algebra but a set. Intuitively, this is the set of behaviours of a computation of type $\underline{B}$. So a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$, because, in a given store $s \in S$ and environment $\rho \in [\![\Gamma]\!]$, it behaves in a certain way. The semantics of types is as follows:

- The behaviour of a computation of type FA is to terminate in a state $s \in S$ returning a value V of type A. So FA denotes $S \times [\![A]\!]$.
- The behaviour of a computation of type $A \to \underline{B}$ is to pop a value of type A, and, depending on the value popped, to behave as a computation of type $\underline{B}$. So $A \to \underline{B}$ denotes $[\![A]\!] \to [\![\underline{B}]\!]$.
- The behaviour of a computation of type $\prod_{i \in I} \underline{B}_i$ is to pop $i \in I$, and, depending on the i popped, to behave as a computation of type $\underline{B}_i$. So $\prod_{i \in I} \underline{B}_i$ denotes $\prod_{i \in I} [\![\underline{B}_i]\!]$.

– A value of type $U\underline{B}$ can be forced in any store $s \in S$, and depending on this store, will behave as a computation of type $\underline{B}$. So $U\underline{B}$ denotes $S \to [\![\underline{B}]\!]$.

The semantic equations for terms are straightforward and we omit them. In behaviour semantics, it is straightforward to formulate a soundness theorem for computations of all types:

Proposition 5. If $s, M \Downarrow s', T$ then $[\![M]\!]s = [\![T]\!]s'$.

and this is proved by induction on $\Downarrow$.

Corollary 2. Let M be a closed ground term. For every $s \in S$, we have $[\![M]\!]s = (s', a)$ iff $s, M \Downarrow s', a$. Hence terms with the same denotation are observationally equivalent.

7. Call-By-Value and Call-By-Name Fragments Of CBPV

7.1. Introduction

In this section, we shall display CBV and CBN as fragments of CBPV. The types of CBV are value types and the types of CBN are computation types. Whereas the CBV boolean and sum types are the same as CBPV, the CBV function type decomposes as

$$A \to_{\mathbf{CBV}} B = U(A \to FB) \tag{4}$$

Operationally, this says that a CBV function is a thunk of a computation that pops an argument and returns an answer. The CBN function type decomposes as

$$\underline{A} \to_{\mathbf{CBN}} \underline{B} \;=\; U\underline{A} \to \underline{B}$$

Operationally, this says that an argument to a CBN function is a thunk. The CBN boolean and sum types decompose as

$$\begin{aligned} \mathtt{bool}_{\mathbf{CBN}} &= F(1+1) \\ \underline{A} +_{\mathbf{CBN}} \underline{A}' &= F(U\underline{A} + U\underline{A}') \\ \underline{A} \to_{\mathbf{CBN}} \underline{B} &= U\underline{A} \to \underline{B} \end{aligned}$$

It is crucial to see that all these decompositions preserve denotational semantics, both for cpos/cppos and, more generally, in the monad/algebra setting. We state this properly in Sect. 7.4.

The types we treat for CBV and CBN are

$$A ::= \ \mathtt{bool} \mid A \to A \mid A + A \tag{5}$$

deferring the treatment of other connectives to Sect. 7.5. For the connectives in (5), the term syntax and big-step semantics are standard e.g. (Winskel, 1993). For recursion, there are several possible formulations, one of which is shown in Fig. 7.

CBN

$$\frac{\Gamma, \mathtt{x} : A \vdash M : A}{\Gamma \vdash \mu\mathtt{x}.M : A} \qquad \frac{M[\mu\mathtt{x}.M/\mathtt{x}] \Downarrow T}{\mu\mathtt{x}.M \Downarrow T}$$

CBV

$$\frac{\Gamma, \mathtt{x} : A, \mathtt{f} : A \to B \vdash M : B}{\Gamma \vdash \mu\mathtt{f}\lambda\mathtt{x}.M : A \to B} \qquad \frac{}{\mu\mathtt{f}\lambda\mathtt{x}.M \Downarrow \mu\mathtt{f}\lambda\mathtt{x}.M}$$

$$\frac{M \Downarrow \mu\mathtt{f}\lambda\mathtt{x}.P \quad N \Downarrow V \quad P[\mu\mathtt{f}\lambda\mathtt{x}.P/\mathtt{f}, V/\mathtt{x}] \Downarrow W}{MN \Downarrow W}$$

Figure 7. Syntax and big-step semantics of recursion in CBN and CBV

7.2. CBN to CBPV

Each CBN term $A_0, \ldots, A_{n-1} \vdash M : B$ translates into a CBPV computation $UA_0^{\mathsf{n}}, \ldots, UA_{n-1}^{\mathsf{n}} \vdash M^{\mathsf{n}} : B^{\mathsf{n}}$. The translation is shown in Fig. 8, and it clearly preserves denotational semantics in the cppo setting, and, more generally, in the monad/algebra setting. However, it does not precisely preserve substitution or big-step semantics; as an example, consider the CBN term let true be x. λy.x.

To obtain such preservation results, we need to work with a *relation* $\mapsto^{\mathsf{n}}$ from CBN terms to CBPV computations. This is defined inductively; there is one rule for each line of Fig. 8 e.g.

$$\frac{}{\mathtt{x} \mapsto^{\mathsf{n}} \mathtt{force\ x}} \qquad \frac{N \mapsto^{\mathsf{n}} N' \quad M \mapsto^{\mathsf{n}} M'}{N`M \mapsto^{\mathsf{n}} (\mathtt{thunk}\ N')`M'}$$

the additional rule

$$\frac{M \mapsto^{\mathsf{n}} M'}{M \mapsto^{\mathsf{n}} \mathtt{force\ thunk}\ M'}$$

C	C^{n} (a computation type)
$\mathtt{bool}$	$F\mathtt{bool}$ i.e. $F(1+1)$
$A+B$	$F(UA^{\mathsf{n}}+UB^{\mathsf{n}})$
$A \to B$	$(UA^{\mathsf{n}}) \to B^{\mathsf{n}}$

$A_0, \ldots, A_{n-1} \vdash M : C$	$UA_0^{\mathsf{n}}, \ldots, UA_{n-1}^{\mathsf{n}} \vdash^{\mathsf{c}} M^{\mathsf{n}} : C^{\mathsf{n}}$
$\mathtt{x}$	$\mathtt{force}\ \mathtt{x}$
$\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N$	$\mathtt{let}\ \mathtt{thunk}\ M^{\mathsf{n}}\ \mathtt{be}\ \mathtt{x}.\ N^{\mathsf{n}}$
$\mathtt{true}$	$\mathtt{return}\ \mathtt{true}$
$\mathtt{false}$	$\mathtt{return}\ \mathtt{false}$
$\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N'$	$M^{\mathsf{n}}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{if}\ \mathtt{z}\ \mathtt{then}\ N^{\mathsf{n}}\ \mathtt{else}\ N'^{\mathsf{n}}$
$\mathtt{inl}\ M$	$\mathtt{return}\ \mathtt{inl}\ \mathtt{thunk}\ M^{\mathsf{n}}$
$\mathtt{inr}\ M$	$\mathtt{return}\ \mathtt{inr}\ \mathtt{thunk}\ M^{\mathsf{n}}$
$\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\}$	$M^{\mathsf{n}}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N^{\mathsf{n}}, \mathtt{inr}\ \mathtt{x}.N'^{\mathsf{n}}\}$
$\lambda\mathtt{x}.M$	$\lambda\mathtt{x}.M^{\mathsf{n}}$
MN	$(\mathtt{thunk}\ N^{\mathsf{n}})\text{`}M^{\mathsf{n}}$
$\mu\mathtt{x}.M$	$\mu.\mathtt{x}.M^{\mathsf{n}}$
$\mathtt{print}\ c.\ M$	$\mathtt{print}\ c.\ M^{\mathsf{n}}$

Figure 8. Translating CBN To CBPV

and the rule for recursion

$$\frac{M \mapsto^{\mathsf{n}} M'}{\mu\mathtt{x}.M \mapsto^{\mathsf{n}} \mu\mathtt{x}.M'}$$

Now substitution is preserved.

Proposition 6. If $M \mapsto^{\mathsf{n}} M'$ and $N \mapsto^{\mathsf{n}} N'$ then $M[N/\mathtt{x}] \mapsto^{\mathsf{n}} M'[\mathtt{thunk}\ N'/\mathtt{x}]$

The relation from CBN to CBPV terms is a bisimulation:

Proposition 7. Suppose M is a closed CBN term, and $M \mapsto^{\mathsf{n}} M'$.

1. If $M \Downarrow_{\mathsf{CBN}} T$ then there exists T' such that $T \mapsto^{\mathsf{n}} T'$ and $M' \Downarrow T'$.

2. If $M' \Downarrow T'$, then there exists T such that $T \mapsto^{\mathsf{n}} T'$ and $M \Downarrow_{\mathsf{CBN}} T$.

Hence the translation reflects observational inequality: if $M^{\mathsf{n}} \lesssim N^{\mathsf{n}}$ then $M \lesssim N$.

7.3. From CBV to CBPV

The translation from CBV to CBPV proceeds in two stages: first from CBV to fine-grain CBV (which we saw in Sect. 2.2), which leaves the types unchanged, and then from fine-grain CBV into CBPV, which decomposes the function type. But, in this paper, we just present the composite translation, and it appears in Fig. 9.

C	C^{v} (a value type)
$\texttt{bool}$	$\texttt{bool}$ i.e. $1+1$
$A+B$	$A^{\mathsf{v}}+B^{\mathsf{v}}$
$A \to B$	$U(A^{\mathsf{v}} \to FB^{\mathsf{v}})$

$A_0,\ldots,A_{n-1} \vdash M : C$	${A_0}^{\mathsf{v}},\ldots,{A_{n-1}}^{\mathsf{v}} \vdash^{\mathsf{c}} M^{\mathsf{prod}} : FC^{\mathsf{v}}$
$\texttt{x}$	$\texttt{return x}$
$\texttt{let x be } M.\ N$	$M^{\mathsf{prod}} \texttt{ to x. } N^{\mathsf{prod}}$
$\texttt{true}$	$\texttt{return true}$
$\texttt{false}$	$\texttt{return false}$
$\texttt{if } M \texttt{ then } N \texttt{ else } N'$	$M^{\mathsf{prod}} \texttt{ to z. if z then } N^{\mathsf{prod}} \texttt{ else } N'^{\mathsf{prod}}$
$\texttt{inl } M$	$M^{\mathsf{prod}} \texttt{ to z. return inl z}$
$\texttt{inr } M$	$M^{\mathsf{prod}} \texttt{ to z. return inr z}$
$\texttt{pm } M \texttt{ as } \{\texttt{inl x}.N, \texttt{inr x}.N'\}$	$M^{\mathsf{prod}} \texttt{ to z. pm z as } \{\texttt{inl x}.N^{\mathsf{prod}}, \texttt{inr x}.N'^{\mathsf{prod}}\}$
$\lambda\texttt{x}.M$	$\texttt{return thunk } \lambda\texttt{x}.M^{\mathsf{prod}}$
MN	$M^{\mathsf{prod}} \texttt{ to f. } N^{\mathsf{prod}} \texttt{ to x. x`(force f)}$
$\texttt{print } c.\ M$	$\texttt{print } c.\ M^{\mathsf{prod}}$
$\mu\texttt{f}\lambda\texttt{x}.M$	$\texttt{return thunk } \mu\texttt{f}\lambda\texttt{x}.M^{\mathsf{prod}}$

$A_0,\ldots,A_{n-1} \vdash V : C$	${A_0}^{\mathsf{v}},\ldots,{A_{n-1}}^{\mathsf{v}} \vdash^{\mathsf{v}} V^{\mathsf{val}} : C^{\mathsf{v}}$
$\texttt{x}$	$\texttt{x}$
$\texttt{true}$	$\texttt{true}$
$\texttt{false}$	$\texttt{false}$
$\texttt{inl } V$	$\texttt{inl } V^{\mathsf{val}}$
$\texttt{inr } V$	$\texttt{inr } V^{\mathsf{val}}$
$\lambda\texttt{x}.M$	$\texttt{thunk } \lambda\texttt{x}.M^{\mathsf{prod}}$
$\mu\texttt{f}\lambda\texttt{x}.M$	$\texttt{thunk } \mu\texttt{f}\lambda\texttt{x}.M^{\mathsf{prod}}$

Figure 9. Translation of CBV types, terms and values

As with the translation from CBN, it does not preserve substitution or big-step semantics. To see this, consider the term $\texttt{let inl true be x. }\lambda\texttt{y.x}$.

So, again, we give a *relation* $\mapsto^{\mathsf{prod}}$ from CBV terms to CBPV terms, and another relation $\mapsto^{\mathsf{val}}$ from CBV values to CBPV values. We can present Fig. 9 by rules such as these:

$$\frac{M \mapsto^{\mathsf{prod}} M' \quad N \mapsto^{\mathsf{prod}} N'}{\mathtt{let\ x\ be}\ M.\ N \mapsto^{\mathsf{prod}} M'\ \mathtt{to\ x.}\ N'} \qquad \frac{M \mapsto^{\mathsf{prod}} M' \ \cdots}{\lambda\mathtt{x}.M \mapsto^{\mathsf{val}} \mathtt{thunk}\ \lambda\mathtt{x}.M'}$$

To these rules we add the following

$$\frac{M \mapsto^{\mathsf{prod}} \mathtt{return}\ V\ \mathtt{to\ x.\ return\ inl\ x}}{M \mapsto^{\mathsf{prod}} \mathtt{return\ inl}\ V} \qquad \frac{M \mapsto^{\mathsf{prod}} \mathtt{return}\ V\ \mathtt{to\ x.\ return\ inr\ x}}{M \mapsto^{\mathsf{prod}} \mathtt{return\ inr}\ V}$$

We have thus defined non-functional relations $\mapsto^{\mathsf{prod}}$ and $\mapsto^{\mathsf{val}}$, and we will show that they commute with substitution and preserve and reflect operational semantics.

Proposition 8. For any producer M, we have $M \mapsto^{\mathsf{prod}} M^{\mathsf{v}}$, and if $M \mapsto^{\mathsf{prod}} N$ then $N = M^{\mathsf{v}}$ is provable in the CBPV equational theory of Fig. 12; similarly for values.

Proposition 9. 1. If $V \mapsto^{\mathsf{val}} V'$ then $V \mapsto^{\mathsf{prod}} \mathtt{return}\ V'$.

2. If $M \mapsto^{\mathsf{prod}} M'$ and $V \mapsto^{\mathsf{val}} V'$ then $M[V/\mathrm{x}] \mapsto^{\mathsf{prod}} M'[V'/\mathrm{x}]$.

3. If $W \mapsto^{\mathsf{val}} W'$ and $V \mapsto^{\mathsf{val}} V'$ then $W[V/\mathrm{x}] \mapsto^{\mathsf{val}} W'[V'/\mathrm{x}]$.

The relation from CBV to CBPV terms is a bisimulation:

Proposition 10. 1. If $M \Downarrow V$ and $M \mapsto^{\mathsf{prod}} M'$, then, for some V', we have $M' \Downarrow \mathtt{return}\ V'$ and $V \mapsto^{\mathsf{val}} V'$.

2. If $M \mapsto^{\mathsf{prod}} M'$ and $M' \Downarrow \mathtt{return}\ V'$, then, for some V, we have $M \Downarrow V$ and $V \mapsto^{\mathsf{val}} V'$.

Hence the translation reflects observational inequality: if $M^{\mathsf{v}} \lesssim N^{\mathsf{v}}$ then $M \lesssim N$.

To prove this, we introduce the following.

Definition 3. 1. In CBV, the following terms are *safe*:

$$\begin{array}{rcl} S & ::= & \mathtt{x} \mid \mathtt{let\ x\ be}\ S.\ S \mid \mathtt{inl}\ S \mid \mathtt{inr}\ S \\ & & \mid \mathtt{pm}\ S\ \mathtt{as}\ \{\mathtt{inl\ x}.S, \mathtt{inr\ x}.S\} \mid \lambda\mathtt{x}.M \end{array}$$

Thus, a term if safe iff, inside it, every application occurs in the scope of a λ.

2. In CBPV the following terms are *safe*:

$$\begin{array}{rl} S ::= & \texttt{return}\ V \mid \texttt{let x be}\ V.\ S \mid S\ \texttt{to x}.\ S \\ & \mid \texttt{pm}\ V\ \texttt{as}\ \{\ldots,(i,\texttt{x}).S_i,\ldots\} \mid \texttt{pm}\ V\ \texttt{as}\ (\texttt{x},\texttt{y}).S \end{array}$$

Thus a term is safe iff, inside it, every application occurs within a value.

Lemma 1. Suppose $A_0,\ldots,A_{n-1} \vdash M \mapsto^{\mathsf{prod}} M' : B$. Then

1. M is safe iff M' is safe.

2. Suppose that M is safe and that $U_0 \mapsto^{\mathsf{val}} U'_0,\ldots,U_{n-1} \mapsto^{\mathsf{val}} U'_{n-1}$.

   – If $M[\overrightarrow{U_i/\texttt{x}_i}] \Downarrow V$, then, for some V', we have $M'[\overrightarrow{U'_i/\texttt{x}_i}] \Downarrow \texttt{return}\ V'$ and $V \mapsto^{\mathsf{val}} V'$.

   – If $M'[\overrightarrow{U'_i/\texttt{x}_i}] \Downarrow \texttt{return}\ V'$, then, for some V, we have $M[\overrightarrow{U_i/\texttt{x}_i}] \Downarrow V$ and $V \mapsto^{\mathsf{val}} V'$.

We prove this by induction on $M \mapsto^{\mathsf{prod}} M'$. Finally we prove Prop. 10 by induction on $M \Downarrow m, V$ (for (1)) and on $M' \Downarrow m, \texttt{return}\ V'$ (for (2)).

7.4. Preservation Of Denotational Semantics

As we stated in the Introduction, the most important results about CBPV are the most trivial ones:

Proposition 11. The translations we have seen, from CBN and from CBV to CBPV preserve cpo semantics, and more generally monad/algebra semantics, up to isomorphism. In other words, the semantics of CBN and CBV obtained from the monad/algebra semantics of CBPV are the monad/algebra semantics described in Sect. 2.

In particular, the monad semantics[3] of $U(A \to FB)$ is an exponential from $[\![A]\!]$ to $T[\![B]\!]$.

Proposition 12. The CBV storage semantics obtained from the storage semantics of CBPV is the traditional one, up to isomorphism, while the CBN storage semantics obtained from the storage semantics of CBPV is that of (O'Hearn, 1993).

[3] This in fact is true for *any* semantics of CBPV—this follows from Prop. 16 below.

In particular, we have

$$\begin{aligned}
[\![A \to_{\mathbf{CBV}} B]\!] &= S \to ([\![A]\!] \to (S \times [\![B]\!])) \\
[\![A \to_{\mathbf{CBN}} B]\!] &= (S \to [\![A]\!]) \to [\![B]\!]
\end{aligned}$$

7.5. More CBN/CBV Connectives

In studying the translation of CBV and CBN into CBPV, we have so far looked at 3 connectives in the source languages: `bool`, $+$, $\to$. We will now look at 2 more, called ζ and ϕ.

The connective ζ is 3-ary. We think of $\zeta(A, B, C)$ as a type containing

- tuples (l, M, M'), where M has type A and M' has type B, and
- tuples (r, M), where M has type C.

We provide ζ with two introduction rules

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M' : B}{\Gamma \vdash (\mathsf{l}, M, M') : \zeta(A, B, C)} \qquad \frac{\Gamma \vdash M : C}{\Gamma \vdash (\mathsf{r}, M) : \zeta(A, B, C)}$$

and with one elimination rule that pattern-matches a tuple

$$\frac{\Gamma \vdash M : \zeta(A, B, C) \quad \Gamma, \mathtt{x} : A, \mathtt{y} : B \vdash N : D \quad \Gamma, \mathtt{x} : C \vdash N' : D}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{(\mathsf{l}, \mathtt{x}, \mathtt{y}).\ N, (\mathsf{r}, \mathtt{x}).\ N'\} : D}$$

The connective ϕ is 5-ary. We think of $\phi(A, B, C, D, E)$ as a type of functions that

- takes a tuple $[\mathsf{l}, M, M']$, where M has type A and M' has type B, to something of type C, and
- takes a tuple $[\mathsf{r}, M]$, where M has type D, to something of type E.

We provide ϕ with one introduction rule, a λ-abstraction that must provide *two* bodies:

$$\frac{\Gamma, \mathtt{x} : A, \mathtt{y} : B \vdash M : C \quad \Gamma, \mathtt{x} : D \vdash M' : E}{\Gamma \vdash \lambda\{[\mathsf{l}, \mathtt{x}, \mathtt{y}].M, [\mathsf{r}, \mathtt{x}].M'\} : \phi(A, B, C, D, E)}$$

and with two elimination rules for application:

$$\frac{\Gamma \vdash M : \phi(A, B, C, D, E) \quad \Gamma \vdash N : A \quad \Gamma \vdash N' : B}{\Gamma \vdash M[\mathsf{l}, N, N'] : C}$$

$$\frac{\Gamma \vdash M : \phi(A, B, C, D, E) \quad \Gamma \vdash N : D}{\Gamma \vdash M[\mathsf{r}, N] : E}$$

It is straightforward to give CBV and CBN operational semantics for all these terms. For recursion, we adapt the CBV formulation in Fig. 7 to allow recursive λ-abstractions of ϕ type.

The translation of these constructs into CBPV is shown in Fig. 10.

CBN

C	C^{n} (a computation type)
$\zeta(A,B,C)$	$F\sum\{\mathsf{l}.(UA^{\mathsf{n}}\times UB^{\mathsf{n}}),\mathsf{r}.UC^{\mathsf{n}}\}$
$\phi(A,B,C,D,E)$	$\prod\{\mathsf{l}.(UA^{\mathsf{n}}\to UB^{\mathsf{n}}\to C^{\mathsf{n}}),\mathsf{r}.(UD^{\mathsf{n}}\to E^{\mathsf{n}})\}$

$A_0,\ldots,A_{n-1}\vdash M:C$	$UA_0^{\mathsf{n}},\ldots,UA_{n-1}^{\mathsf{n}}\vdash^{\mathsf{c}} M^{\mathsf{n}}:C^{\mathsf{n}}$
(l,M,M')	$\mathtt{return}\ (\mathsf{l},(\mathtt{thunk}\ M^{\mathsf{n}},\mathtt{thunk}\ M'^{\mathsf{n}}))$
(r,M)	$\mathtt{return}\ (\mathsf{r},\mathtt{thunk}\ M^{\mathsf{n}})$
$\mathtt{pm}\ M\ \mathtt{as}\ \{(\mathsf{l},\mathtt{x},\mathtt{y}).\ N,(\mathsf{r},\mathtt{x}).\ N'\}$	$M^{\mathsf{n}}\ \mathtt{to\ z.\ pm\ z\ as}\ \{(\mathsf{l},(\mathtt{x},\mathtt{y})).N^{\mathsf{n}},(\mathsf{r},\mathtt{x}).N'^{\mathsf{n}}\}$
$\lambda\{[\mathsf{l},\mathtt{x},\mathtt{y}].M,[\mathsf{r},\mathtt{x}].M'\}$	$\lambda\{\mathsf{l}.\lambda\mathtt{x}.\lambda\mathtt{y}.M^{\mathsf{n}},\mathsf{r}.\lambda\mathtt{x}.M'^{\mathsf{n}}\}$
$M[\mathsf{l},N,N']$	$\mathtt{thunk}\ N'^{\mathsf{n}}\text{`}\mathtt{thunk}\ N^{\mathsf{n}}\text{`}\mathsf{l}\text{`}M^{\mathsf{n}}$
$M[\mathsf{r},N]$	$\mathtt{thunk}\ N^{\mathsf{n}}\text{`}\mathsf{r}\text{`}M^{\mathsf{n}}$

CBV

C	C^{v} (a value type)
$\zeta(A,B,C)$	$\sum\{\mathsf{l}.(A^{\mathsf{n}}\times B^{\mathsf{n}}),\mathsf{r}.C^{\mathsf{n}}\}$
$\phi(A,B,C,D,E)$	$U\prod\{\mathsf{l}.(A^{\mathsf{n}}\to B^{\mathsf{n}}\to FC^{\mathsf{n}}),\mathsf{r}.(D^{\mathsf{n}}\to FE^{\mathsf{n}})\}$

$A_0,\ldots,A_{n-1}\vdash M:C$	${A_0}^{\mathsf{v}},\ldots,{A_{n-1}}^{\mathsf{v}}\vdash^{\mathsf{c}} M^{\mathsf{prod}}:FC^{\mathsf{v}}$
(l,M,M')	$M^{\mathsf{v}}\ \mathtt{to\ x.}\ M'^{\mathsf{v}}\ \mathtt{to\ y.\ return}\ (\mathsf{l},(\mathtt{x},\mathtt{y}))$
(r,M)	$M^{\mathsf{v}}\ \mathtt{to\ x.\ return}\ (\mathsf{r},\mathtt{x})$
$\mathtt{pm}\ M\ \mathtt{as}\ \{(\mathsf{l},\mathtt{x},\mathtt{y}).\ N,(\mathsf{r},\mathtt{x}).\ N'\}$	$M^{\mathsf{v}}\ \mathtt{to\ z.\ pm\ z\ as}\ \{(\mathsf{l},(\mathtt{x},\mathtt{y})).N^{\mathsf{v}},(\mathsf{r},\mathtt{x}).N'^{\mathsf{v}}\}$
$\lambda\{[\mathsf{l},\mathtt{x},\mathtt{y}].M,[\mathsf{r},\mathtt{x}].M'\}$	$\mathtt{return\ thunk}\ \lambda\{\mathsf{l}.\lambda\mathtt{x}.\lambda\mathtt{y}.M^{\mathsf{v}},\mathsf{r}.\lambda\mathtt{x}.M'^{\mathsf{v}}\}$
$M[\mathsf{l},N,N']$	$\mathtt{return}\ M^{\mathsf{v}}\ \mathtt{to\ f.\ return}\ N^{\mathsf{v}}\ \mathtt{to\ x.}$ $\mathtt{return}\ N'^{\mathsf{v}}\ \mathtt{to\ y.\ y}\text{`}\mathtt{x}\text{`}\mathtt{force\ f}$
$M[\mathsf{r},N]$	$\mathtt{return}\ M^{\mathsf{v}}\ \mathtt{to\ f.\ return}\ N^{\mathsf{v}}\ \mathtt{to\ x.}$ $\mathtt{x}\text{`}\mathtt{force\ f}$
$\mu\mathtt{f}\lambda\{[\mathsf{l},\mathtt{x},\mathtt{y}].M,[\mathsf{r},\mathtt{x}].M'\}$	$\mathtt{return\ thunk}\ \mu\mathtt{f}.\lambda\{\mathsf{l}.\lambda\mathtt{x}.\lambda\mathtt{y}.M^{\mathsf{v}},\mathsf{r}.\lambda\mathtt{x}.M'^{\mathsf{v}}\}$

$A_0,\ldots,A_{n-1}\vdash V:C$	${A_0}^{\mathsf{v}},\ldots,{A_{n-1}}^{\mathsf{v}}\vdash^{\mathsf{v}} V^{\mathsf{val}}:C^{\mathsf{v}}$
(l,V,V')	$(\mathsf{l},(V^{\mathsf{val}},V'^{\mathsf{val}}))$
(r,V)	$(\mathsf{r},V^{\mathsf{val}})$
$\lambda\{[\mathsf{l},\mathtt{x},\mathtt{y}].M,[\mathsf{r},\mathtt{x}].M'\}$	$\mathtt{thunk}\ \lambda\{\mathsf{l}.\lambda\mathtt{x}.\lambda\mathtt{y}.M^{\mathsf{v}},\mathsf{r}.\lambda\mathtt{x}.M'^{\mathsf{v}}\}$
$\mu\mathtt{f}\lambda\{[\mathsf{l},\mathtt{x},\mathtt{y}].M,[\mathsf{r},\mathtt{x}].M'\}$	$\mathtt{thunk}\ \mu\mathtt{f}\lambda\{\mathsf{l}.\lambda\mathtt{x}.\lambda\mathtt{y}.M^{\mathsf{v}},\mathsf{r}.\lambda\mathtt{x}.M'^{\mathsf{v}}\}$

Figure 10. Translating ζ and ϕ from CBN and CBV into CBPV

More generally, suppose $\{n_i\}_{i \in I}$ is a finite family of natural numbers. Then this gives rise to a tuple connective with arity $\sum_{i \in I} n_i$, and to a function connective with arity $\sum_{i \in I}(n_i + 1)$. For example, the tuple connective ζ and the function connective ϕ are described by the family

$$I = \{\mathsf{l}, \mathsf{r}\} \qquad n_{\mathsf{l}} = 2 \qquad n_{\mathsf{r}} = 1$$

Henceforth, we shall assume the CBV and CBN source languages to be equipped with these connectives for *every* finite family of naturals. All the connectives we have seen so far, viz. $\texttt{bool}, +, \rightarrow, \zeta, \phi$, as well as the n-ary function types mentioned in Sect. 2.2, are included among these. Also included are two binary products, one of which is a tuple connective and the other a function connective. These are not, in fact, isomorphic in either CBV or CBN.

We translate all these connectives into CBPV the same way we translated ζ and ϕ, and adapt the proofs of Prop. 7–12 accordingly.

Proposition 13. 1. The translation from CBN (with all tuple and function connectives) to CBPV is fully abstract, i.e. $M \lesssim N$ iff $M^{\mathsf{n}} \lesssim N^{\mathsf{n}}$.

2. The translation from CBV (with all tuple and function connectives) to CBPV is fully abstract, i.e. $M \lesssim N$ iff $M^{\mathsf{v}} \lesssim N^{\mathsf{v}}$.

We omit the proof of this, which uses a reverse translation from CBPV to CBN and from CBPV to CBV. The details can be found in (Levy, 2004).

It should be noted that all these results adapt to each of the effects we consider: divergence, printing and storage.

7.6. Untyped Languages

The simulation results Prop. 6–10 do not make any use of types, and could be transferred to an untyped setting. However, this appears to be of little interest. After all, an essential part of the motivation we presented for CBPV was the idea of observing convergence at ground type only, and this makes no sense in an untyped language. Instead, to validate the η-law in the absence of ground types, the traditional observation in untyped CBN λ-calculus is *reduction to head normal form.* That is unsuited to typed languages, e.g. it would distinguish the CBN terms $\lambda\texttt{x.x}$ and $\lambda\texttt{x.diverge}$ of type $1 \rightarrow 1$, even though they have the same denotation.

For untyped languages, moreover, the fact that terms can get stuck complicates both operational and denotational semantics. The latter is further complicated by the need to solve domain/predomain equations, and, in the case of untyped CBN λ-calculus, to use a non-bifree solution[4], because the bifree solution of $D \cong D \to D$ is trivial.

8. Complex Values and the CBPV Equational Theory

In this section, we give the CBPV equational theory, which will allow us to prove correspondence with the categorical semantics, and to relate CBPV to Filinski's Effect-PCF (Filinski, 1996).

The CBPV typing rules presented in Fig. 2 allow pattern-matching into computations, but not into values. As stated for fine-grain CBV in Remark 1, this keeps the operational semantics simple. But in this section, we add complex values to CBPV, both for categorical correspondence, and to make the relationship with Effect-PCF precise. Complex values (terms that are denotationally values but need to be evaluated) are given by the following rules. It is clear how to interpret

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

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

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

Figure 11. Complex Values

these constructs in all of our denotational models.

We define the CBPV equational theory to be that presented in Fig. 12. The laws given for `print` and `diverge` are of course effect-specific, but there are similar laws for other effects.

The push/pop reading of CBPV sheds light on may of these laws. For example, the β-law for functions says: "if we push V, then pop x, then do M, that is the same as doing M with x bound to V". Similarly the η-law for functions says: "if we pop x, then push x, then do M which ignores x, that is the same as doing M".

[4] A semantics using bifree solutions appears in (Streicher and Reus, 1998), however.

R ranges over both values and computations. We omit the assumptions necessary to make each equation well-typed. Given a term $\Gamma \vdash R : B$ we write ${}^{\mathtt{x}}R$ for the weakened term in the context $\Gamma, \mathtt{x} : A$ where A is some suitable type. This implies that $\mathtt{x}$ is not in Γ, because the identifiers in a context must be distinct. We thereby obviate the need for the traditional $\mathtt{x} \notin \mathsf{FV}(R)$ conditions.

β-laws

$$
\begin{array}{ccc}
\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ R & = & R[V/\mathtt{x}] \\
(\mathtt{return}\ V)\ \mathtt{to}\ \mathtt{x}.\ M & = & M[V/\mathtt{x}] \\
\mathtt{force}\ \mathtt{thunk}\ M & = & M \\
\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).R_i,\ldots\} & = & R_{\hat{\imath}}[V/\mathtt{x}] \\
\mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).R & = & R[V/\mathtt{x}, V'/\mathtt{y}] \\
\hat{\imath}`\lambda\{\ldots,i.M_i,\ldots\} & = & M_{\hat{\imath}} \\
V`\lambda\mathtt{x}.M & = & M[V/\mathtt{x}]
\end{array}
$$

η-laws

$$
\begin{array}{ccc}
M & = & M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{return}\ \mathtt{x} \\
V & = & \mathtt{thunk}\ \mathtt{force}\ V \\
R[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).\ {}^{\mathtt{x}}R[(i,\mathtt{x})/\mathtt{z}],\ldots\} \\
R[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\ {}^{\mathtt{xy}}R[(\mathtt{x},\mathtt{y})/\mathtt{z}] \\
M & = & \lambda\{\ldots,i.i`M,\ldots\} \\
M & = & \lambda\mathtt{x}.(\mathtt{x}`\ {}^{\mathtt{x}}M)
\end{array}
$$

sequencing laws

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

`print` laws

$$
\begin{array}{ccc}
(\mathtt{print}\ c.\ M)\ \mathtt{to}\ \mathtt{x}.\ N & = & \mathtt{print}\ c.\ (M\ \mathtt{to}\ \mathtt{x}.\ N) \\
\mathtt{print}\ c.\ \lambda\{\ldots,i.M_i,\ldots\} & = & \lambda\{\ldots,i.(\mathtt{print}\ c.\ M),\ldots\} \\
\mathtt{print}\ c.\ \lambda\mathtt{x}.M & = & \lambda\mathtt{x}.(\mathtt{print}\ c.\ M)
\end{array}
$$

`diverge` laws

$$
\begin{array}{ccc}
\mathtt{diverge}\ \mathtt{to}\ \mathtt{x}.\ N & = & \mathtt{diverge} \\
\mathtt{diverge} & = & \lambda\{\ldots,i.\mathtt{diverge},\ldots\} \\
\mathtt{diverge} & = & \lambda\mathtt{x}.\mathtt{diverge}
\end{array}
$$

Figure 12. CBPV equations

Proposition 14. (soundness of equational theory) The equations in Fig. 12 are validated by our denotational models for divergence, printing and storage, and hence are observational equivalences in the presence of any one of these effects.

Proposition 15. (eliminability of complex values) 1. Every computation M is provably equal to a complex-value-free computation M'.

2. Every *closed* value V is provably equal to a complex-value-free closed value V'.

3. There exists a non-closed value V which is not provably equal to any complex-value-free value.

Proof. By induction over terms we define in Fig. 13

– for each computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$, a complex-value-free computation $\Gamma \vdash^{\mathsf{c}} \tilde{M} : \underline{B}$ such that $M = \tilde{M}'$ is provable

– for each value $\Gamma \vdash^{\mathsf{v}} V : A$, a function taking each complex-value-free computation $\Gamma, \mathtt{v} : A \vdash^{\mathsf{c}} N : \underline{B}$ to a complex-value-free computation $\Gamma \vdash^{\mathsf{c}} N[V /\!\!/ \mathtt{v}] : \underline{B}$ such that $N[V /\!\!/ \mathtt{v}] = N[V/\mathtt{v}]$ is provable.

We have proved (1) taking M' to be $\tilde{M}$. Next, for any value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : A$ possibly containing complex values, we define, by induction on V in Fig. 13, a function $\tilde{V}$ taking a sequence $W_0, \ldots, W_{n-1}$ of complex-value-free closed values $\vdash^{\mathsf{v}} W_i : A_i$ to a complex-value-free closed value $\tilde{V}(W_0, \ldots, W_{n-1})$ such that

$$\tilde{V}(W_0, \ldots, W_{n-1}) = V[\overrightarrow{W_i/\mathtt{x}_i}]$$

is provable. The special case $n = 0$ gives (2).

For (3) we take V to be $\mathtt{x} : 0 \times 0 \vdash^{\mathsf{v}} \mathtt{pm\ x\ as}\ (\mathtt{y}, \mathtt{z}).\mathtt{y} : 0$. The result is then obvious.

Prop. 15(1) enables us to “evaluate” a computation with closed values, by first removing the complex values and then evaluating. But it should be noted that the algorithm for removal of complex values that we gave in the proof involves some arbitrary choices, and is certainly not canonical. This is essentially the problem with complex values, from the operational perspective: they detract from the rigid sequential nature of the language, because they can be evaluated at any time.

V	$N[V/\!/\mathtt{v}]$
$\mathtt{x}$	$N[\mathtt{x}/\mathtt{v}]$
$\mathtt{let}\ W\ \mathtt{be}\ \mathtt{x}.\ U$	$(\mathtt{let}\ \mathtt{w}\ \mathtt{be}\ \mathtt{x}.\ N[U/\!/\mathtt{v}])[W/\!/\mathtt{w}]$
$(\hat{\imath}, W)$	$(\mathtt{let}\ (\hat{\imath}, \mathtt{w})\ \mathtt{be}\ \mathtt{v}.\ N)[W/\!/\mathtt{w}]$
$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).U_i, \ldots\}$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N[U_i/\!/\mathtt{v}], \ldots\})[W/\!/\mathtt{w}]$
(W, W')	$(\mathtt{let}\ (\mathtt{w}, \mathtt{x})\ \mathtt{be}\ \mathtt{v}.\ N)[W/\!/\mathtt{w}][W'/\!/\mathtt{x}]$
$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).U$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N[U/\!/\mathtt{v}])[W/\!/\mathtt{w}]$
$\mathtt{thunk}\ M$	$N[\mathtt{thunk}\ \tilde{M}/\mathtt{v}]$

M	$\tilde{M}$
$\mathtt{let}\ W\ \mathtt{be}\ \mathtt{x}.\ N$	$(\mathtt{let}\ \mathtt{w}\ \mathtt{be}\ \mathtt{x}.\ \tilde{N})[W/\!/\mathtt{w}]$
$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i, \ldots\}$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).\tilde{N}_i, \ldots\})[W/\!/\mathtt{w}]$
$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).\tilde{N})[W/\!/\mathtt{w}]$
$\lambda\{\ldots, (i, \mathtt{x}).N_i, \ldots\}$	$\lambda\{\ldots, (i, \mathtt{x}).\tilde{N}_i, \ldots\}$
$\hat{\imath}`N$	$\hat{\imath}`\tilde{N}$
$\lambda\mathtt{x}.N$	$\lambda\mathtt{x}.\tilde{N}$
$V`N$	$(\mathtt{v}`\tilde{N})[V/\!/\mathtt{v}]$
$\mathtt{return}\ V$	$(\mathtt{return}\ \mathtt{v})[V/\!/\mathtt{v}]$
$N\ \mathtt{to}\ \mathtt{x}.\ P$	$\tilde{N}\ \mathtt{to}\ \mathtt{x}.\ \tilde{P}$
$\mathtt{force}\ V$	$(\mathtt{force}\ \mathtt{v})[V/\!/\mathtt{v}]$
$\mu\mathtt{x}.M$	$\mu\mathtt{x}.\tilde{M}$
$\mathtt{print}\ c.\ N$	$\mathtt{print}\ c.\ \tilde{N}$

V	$\tilde{V}(\overrightarrow{W_i})$
$\mathtt{x}_i$	W_i
$\mathtt{let}\ W\ \mathtt{be}\ \mathtt{x}.\ U$	$\tilde{U}(\overrightarrow{W_i}, \tilde{W}(\overrightarrow{W_i}))$
$(\hat{\imath}, W)$	$(\hat{\imath}, \tilde{W}(\overrightarrow{W_i}))$
$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).U_i, \ldots\}$	$\tilde{U}_{\hat{\imath}}(\overrightarrow{W_i}, V')$ where $\tilde{W}(\overrightarrow{W_i}) = (\hat{\imath}, V')$
(W, W')	$(\tilde{W}(\overrightarrow{W_i}), \tilde{W}'(\overrightarrow{W_i}))$
$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).U$	$\tilde{U}(\overrightarrow{W_i}, V', V'')$ where $\tilde{W}(\overrightarrow{W_i}) = (V', V'')$
$\mathtt{thunk}\ M$	$\mathtt{thunk}\ \tilde{M}[\overrightarrow{W_i/\mathtt{x}_i}]$

Figure 13. Definitions used in proof of Prop. 15

9. Every Model Is Equivalent To An Algebra Model

In Sect. 3, we saw how to model CBPV in an algebra-building structure, where all exponentials to carriers are required to exist. But this requirement is too strong. If there is a family of algebras containing all free algebras and closed under exponentiation and finite product, then it suffices to require exponentials to carriers of algebras in this family. We make this precise as follows.

Definition 4. A *CBPV algebra-family* consists of a distributive category $\mathcal{C}$ equipped with a strong monad T and a (not necessarily small) family of T-algebras $\{K\underline{Y}\}_{\underline{Y}\in\mathfrak{J}}$—we write $K\underline{Y} = (U\underline{Y}, \beta\underline{Y})$—together with

free algebras for each $X \in \mathsf{ob}\,\mathcal{C}$, an index $FA \in \mathfrak{J}$ mapped by K to the free algebra on A

exponential algebras for each $X \in \mathsf{ob}\,\mathcal{C}$ and index $\underline{Y} \in \mathfrak{J}$, an exponential E from A to $U\underline{Y}$ in $\mathcal{C}$, and an index $X \to \underline{Y} \in \mathfrak{J}$ mapped by K to the exponential algebra from X to $K\underline{Y}$ constructed using E

product algebras for each finite family $\{\underline{Y}_i\}_{i\in I}$ of indices, a product P for $\{U\underline{Y}_i\}_{i\in I}$ in $\mathcal{C}$, and an index $\prod_{i\in I}\underline{Y}_i \in \mathfrak{J}$ mapped by K to the product algebra of $\{K\underline{Y}_i\}_{i\in I}$ constructed using P.

Clearly this gives us a model of CBPV, where a computation type denotes an index in $\mathfrak{J}$, and a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $U[\![\underline{B}]\!]$. Again, `thunk` and `force` are invisible in all such models.

Remark 2. It is possible to define a "weak" notion of CBPV algebra-family where all the algebra equations in Def. 4 are replaced by algebra isomorphisms (no coherence conditions required). But it can be shown that every such weak model is equivalent to a model in the sense of Def. 4.

We now show that every model of CBPV is an algebra-family. More precisely, we construct a category of CBPV models and a category of CBPV algebra-families and prove them equivalent. Strictly speaking, these should be 2-categories, but to skirt 2-categorical issues, we fix the object structure[5].

Definition 5. 1. A *CBPV object structure* τ is a (not necessarily small) algebra for the 2-sorted signature defining CBPV types. Thus it consists of 2 sets $\mathsf{valtypes}\,\tau$ of *val-objects* and $\mathsf{comptypes}\,\tau$ of *comp-objects*, equipped with a binary operation $\times$ on $\mathsf{valtypes}\,\tau$, and with similar operations for all the other CBPV connectives.

[5] The price we pay for this is that the method is not at all robust. We leave to future work the development of a robust 2-categorical treatment, where structure is preserved only up to isomorphism. But this is a general concern in the categorical semantics of simply typed languages, not specific to CBPV.

2. We write $\mathbf{RestrAlg}_\tau$ for the category of CBPV algebra-families with object structure τ, where morphisms are identity on both val-objects and comp-objects, and preserve all structure on the nose.

Defining the category of CBPV models, purely from the equational theory, is more difficult. The following is a method formulated independently in (Jeffrey, 1999; Levy, 1996), which is applicable to many simply typed calculi.

Definition 6. Let τ be a CBPV object structure.

1. A *τ-sequent* is either

$$A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} B \qquad \text{or} \qquad A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} \underline{B}$$

   where $A_0, \ldots, A_{n-1}$ and B are value objects in τ and $\underline{B}$ is a computation object in τ.

2. A *τ-signature* is a function from τ-sequents to sets.

3. We write Sig_τ for the category of τ-signatures, where a morphism from s to s' provides a function from $s(Q)$ to $s'(Q)$ for each τ-sequent Q.

It is clear that, to model CBPV, one must first give a CBPV object structure τ and then a τ-multigraph s. This much allows us to interpret types and judgements, although it still remains to describe the semantics of term constructors.

Definition 7. Let τ be a CBPV object structure. We define a monad $\mathcal{T}$ on Sig_τ as follows. Let s be a τ-signature. We inductively define another τ-signature called the *terms built from the signature s*, using the rules of Fig. 2 and Fig. 11 together with the rules

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

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

We define $\mathcal{T}s$ to be this signature (mapping each sequent to the terms inhabiting it) quotiented by the congruence generated by the equations of Fig. 12. The unit ηs takes each operation $f \in s\,(A_0, \ldots, A_{r-1} \vdash^{\mathsf{v}} B)$ to $f(\mathtt{x}_0, \ldots, \mathtt{x}_{r-1})$, and similarly for values. The multiplication μs is defined by induction over terms in $\mathcal{T}^2 s$. In particular, it maps $M(V_0, \ldots, V_{n-1})$, where M is an term in $\mathcal{T}s$ and hence an operation in $\mathcal{T}^2 s$, to $M[\overrightarrow{(\mu s)V_j/\mathtt{x}_j}]$, and it preserves all other term constructors.

Definition 8. A *direct model of CBPV* consists of a CBPV object structure τ together with an algebra (s, θ) for the monad $\mathcal{T}$ on Sig_τ. If τ is a CBPV object structure, we write $\mathbf{Direct}\tau$ for the category of $\mathcal{T}$-algebras and algebra homomorphisms.

We can now state our main theorem.

Proposition 16. Let τ be a CBPV object structure. The categories $\mathbf{Direct}_\tau$ and $\mathbf{RestrAlg}_\tau$ are equivalent.

We omit the detailed proof of this, but give an overview. Mapping $\mathbf{RestrAlg}_\tau$ to $\mathbf{Direct}_\tau$ essentially says that a CBPV algebra family gives a model of CBPV, validating all the laws. In the other direction, if we have a model of CBPV, then the semantics of values is a model of the simply typed λ-calculus with $\times$ and $\sum$ types, hence a distributive category $\mathcal{C}$. We obtain a monad by setting T to be UF, and the unit of the monad ηA is given by

$$\mathtt{x} : A \vdash^{\mathsf{v}} \mathtt{thunk\ return\ x} : UFA$$

For each comp-object $\underline{B}$, the algebra $K\underline{B}$ is defined to have carrier $U\underline{B}$ and structure $\beta\underline{B}$, defined by the term

$$\mathtt{x} : UFU\underline{B} \vdash^{\mathsf{v}} \mathtt{thunk}\ (\mathtt{force\ x\ to\ y.\ return\ y}) : U\underline{B}$$

and this determines the multiplication: $\mu A = \beta FA$.

As an instance of this construction, the behaviour semantics of storage (Sect. 6.2) is equivalent to a CBPV algebra family, which is a sub-model of the algebra semantics in Sect. 6.1. This fact enables us to deduce Prop. 4 from Prop. 5.

10. Comparison With Filinski's Monadic Metalanguage and Marz' SFPL

Having treated complex values in some detail, we are in a position to look closely at the relationship between CBPV, Filinski's version of the monadic metalanguage (called Effect-PCF) (Filinski, 1996) and SFPL (Marz, 2000) (a variant of the earlier language SFL (Marz, 1998)). We give the types of both languages in Fig. 14 and the terms of Effect-PCF in Fig. 15. We modify syntax slightly to agree with CBPV (in particular, using M `to x.` N for sequencing), and we exclude the natural number type and recursive types.

It is quite easy to see that if we take the types of CBPV and erase U, so that computation types are a subset of value types, we obtain

Effect-PCF Types

value types	$A ::=$	$\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A$
computation types	$\underline{B} ::=$	$TA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B}$

SFPL Types (NB Marz uses the term “computational type” instead of “value type”.)

value types	$A ::=$	$\underline{B}_\perp \mid \bigoplus_{i\in I} A_i \mid 1_\otimes \mid A \otimes A$
computation types	$\underline{B} ::=$	$A \mid \prod_{i\in I} \underline{B}_i \mid A \multimap \underline{B}$

Figure 14. Types of Effect-PCF and SFPL

the types of Effect-PCF. On the other hand, if we erase F, so that value types are a subset of computation types, we obtain the types of SFPL. This erasure can be explained by the denotational semantics each author was considering:

– Filinski was considering *carrier semantics* where a computation type $\underline{B}$ denotes a carrier of an algebra, rather than the whole algebra, and therefore U is invisible.

– Marz was considering *lifted cpo semantics* where a value type A denotes a pointed cpo, the lift of what A denotes in our cpo semantics. Thus a computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$ denotes a strict function from $[\![A_0]\!] \otimes \cdots \otimes [\![A_{n-1}]\!]$ to $[\![\underline{B}]\!]$. This semantics uses tensor product and coalesced sum of cppos, and strict function spaces. Most importantly, F is invisible.

The translation $\overline{-}$ from CBPV value (resp. computation) types to Effect-PCF value (resp. computation) types are defined by induction:

$$\begin{aligned} \overline{U\underline{B}} &= \overline{\underline{B}} \\ \overline{FA} &= T\overline{A} \end{aligned}$$

and all the other clauses are trivial. The translations $\widetilde{-}$ from Effect-PCF value types to CBPV value types, and $\widehat{-}$ from Effect-PCF computation types to CBPV computation types are defined by mutual induction:

$$\begin{aligned} \widetilde{\beta} &= U\widehat{\beta} \\ \widehat{T\alpha} &= F\widetilde{\alpha} \end{aligned}$$

and all the other clauses are trivial. It is obvious that these define a bijection. In the same way, we can define a bijection between CBPV types and SFPL types.

Primitives

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash \mathtt{x} : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma, \mathtt{x} : A \vdash N : B}{\Gamma \vdash \mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N : B}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathtt{return}\ M : TA} \qquad \frac{\Gamma \vdash M : TA \quad \Gamma, \mathtt{x} : A \vdash N : TB}{\Gamma \vdash M\ \mathtt{to}\ \mathtt{x}.\ N : TB}$$

$$\frac{\Gamma \vdash M : A_{\hat{\imath}}}{\Gamma \vdash (\hat{\imath}, M) : \sum_{i \in I} A_i} \qquad \frac{\Gamma \vdash M : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash N_i : B \quad \cdots_{\ i \in I}}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i, \ldots\} : B}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M' : A'}{\Gamma \vdash (M, M') : A \times A'} \qquad \frac{\Gamma \vdash M : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash N : B}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N : B}$$

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

$$\frac{\Gamma, \mathtt{x} : A \vdash M : \underline{B}}{\Gamma \vdash \lambda \mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash N : A \to \underline{B}}{\Gamma \vdash M\text{`}N : \underline{B}}$$

Derived

$$\frac{\Gamma \vdash M : TA \quad \Gamma, \mathtt{x} : A \quad N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.^{\underline{B}} N : \underline{B}}$$

This is defined by induction on $\underline{B}$.

$$\begin{array}{rcl} M\ \mathtt{to}\ \mathtt{x}.^{TB}\ N & = & M\ \mathtt{to}\ \mathtt{x}.\ N \\ M\ \mathtt{to}\ \mathtt{x}.^{\prod_{i \in I} \underline{B}_i}\ N & = & \lambda\{\ldots, i.(M\ \mathtt{to}\ \mathtt{x}.^{\underline{B}_i}\ (i\text{`}N)), \ldots\} \\ M\ \mathtt{to}\ \mathtt{x}.^{A \to \underline{B}}\ N & = & \lambda \mathtt{y}.(M\ \mathtt{to}\ \mathtt{x}.^{\underline{B}}\ (\mathtt{y}\text{`}N)) \end{array}$$

Figure 15. Terms of Effect-PCF (slightly modified)

Remark 3. Filinski provided type recursion for value types but not computation types. Indeed, computation type identifiers and recursive computation types cannot be allowed in Effect-PCF because they prevent the inductive definition of generalized sequencing. We have seen that CBPV avoids this problem.

Proceeding to terms, we define a translation $\overline{-}$ taking

– a value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : B$ in CBPV with complex values to a term $\overline{A_0}, \ldots, \overline{A_{n-1}} \vdash \overline{V} : \overline{B}$ in Effect-PCF

– a computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$ in CBPV with complex values to a term $\overline{A_0}, \ldots, \overline{A_{n-1}} \vdash \overline{M} : \overline{\underline{B}}$ in Effect-PCF

by induction:

$$\begin{aligned} \overline{\mathtt{thunk}\ M} &= \overline{M} \\ \overline{\mathtt{force}\ V} &= \overline{V} \end{aligned}$$

and all the other clauses are trivial.

In the opposite direction, we define a translation $\widetilde{-}$ taking a term $A_0, \ldots, A_{n-1} \vdash M : B$ in Effect-PCF to a value $\widetilde{A_0}, \ldots, \widetilde{A_{n-1}} \vdash^{\mathsf{v}} \widetilde{M} : \widetilde{B}$, by induction:

$$\begin{aligned} \widetilde{\mathtt{x}} &= \mathtt{x} \\ \widetilde{\mathtt{let}\ M\ \mathtt{be\ x.}\ N} &= \mathtt{let}\ \widetilde{M}\ \mathtt{be\ x.}\ \widetilde{N} \\ \widetilde{(\hat{\imath}, M)} &= (\hat{\imath}, \widetilde{M}) \\ \widetilde{\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i\}} &= \mathtt{pm}\ \widetilde{M}\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).\widetilde{N_i}, \ldots\} \\ \widetilde{(M, M')} &= (\widetilde{M}, \widetilde{M'}) \\ \widetilde{\mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).\ N} &= \mathtt{pm}\ \widetilde{M}\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).\ \widetilde{N} \\ \widetilde{\mathtt{return}\ M} &= \mathtt{thunk}\ \mathtt{return}\ \widetilde{M} \\ \widetilde{M\ \mathtt{to\ x.}\ N} &= \mathtt{thunk}\ (\mathtt{force}\ \widetilde{M}\ \mathtt{to\ x.\ force}\ \widetilde{N}) \\ \widetilde{\lambda \mathtt{x}.M} &= \mathtt{thunk}\ \lambda \mathtt{x}.\mathtt{force}\ \widetilde{M} \\ \widetilde{N`M} &= \mathtt{thunk}\ (\widetilde{N}`\mathtt{force}\ \widetilde{M}) \\ \widetilde{\lambda\{\ldots, i.M_i, \ldots\}} &= \mathtt{thunk}\ \lambda\{\ldots, i.\mathtt{force}\ \widetilde{M_i}, \ldots\} \\ \widetilde{\hat{\imath}`M} &= \hat{\imath}`\widetilde{M} \end{aligned}$$

It is easy to see that these translations are inverse up to `thunk force`. They both preserve (and hence reflect) provable equality in the associated equational theory.

11. Conclusions

We summarize the advances represented by CBPV.

Firstly, the explicit writing of U allows us to give a compositional account of CBN, because a computation type denotes an algebra.

Secondly, CBPV makes explicit the thunking isomorphism, which is invisible from the monadic viewpoint, but apparent in the behaviour semantics of Sect. 6.2.

Thirdly, we see a simple decomposition of CBN and CBV models for the first time. In particular, O'Hearn's behaviour semantics of

CBN (O'Hearn, 1993), where $\underline{A} \to_{\mathbf{CBN}} \underline{B}$ denotes $(S \times [\![\underline{A}]\!]) \to [\![\underline{B}]\!]$ previously appeared strange, but now can be understood using the decomposition of $\underline{A} \to_{\mathbf{CBN}} \underline{B}$ into $U\underline{A} \to \underline{B}$ and the behaviour semantics of CBN. A similar example is the continuation semantics of (Streicher and Reus, 1998), although we have not treated it in this paper.

Fourthly, we have a straightforward operational semantics for CBPV (unlike Effect-PCF, but like MIL-lite), and the translations from CBN and CBV into it are fully abstract. Admittedly, the operational semantics is defined only for complex-value-free terms, but we proved that every computation is equal (in the theory) to one of this form.

Fifthly, we have a machine reading of CBPV (the CK-machine) that makes it clear why a function type should be regarded as a computation type, a classification that was present in Effect-PCF but not understood in computational terms.

As stated in Sect. 1.1, this paper is an introduction to CBPV, not an exhaustive study. In particular, the relationship between CBPV and adjunctions (Levy, 2003; Levy,) is not investigated in this paper. However, in the particular models we have studied, it is quite apparent that U and F represent an adjunctions.

- The monad/algebra semantics uses an Eilenberg-Moore adjunction between $\mathcal{C}$ and $\mathcal{C}^T$ (algebras and algebra homomorphisms).
- The behaviour semantics uses the adjunction between **Set** and **Set** with left adjoint $S \times -$ and right adjoint $S \to -$.

We leave to future work the development of this theory—much more can be found in (Levy, 2004). Furthermore, it remains to compare this work to the line of research in (Laurent, 1999; Selinger, 2001); this is closely related to continuation semantics, which we have not included in this paper.

ACKNOWLEDGEMENTS

I am grateful to Peter O'Hearn and numerous other people for discussion on this material.

References

Benton, N., J. Hughes, and E. Moggi: 2000, 'Monads and Effects'. Lecture notes for Int. Summer School on Applied Semantics, APPSEM'00, Caminha, Portugal, 9–15 Sept. 2000.

Benton, N. and A. Kennedy: 2000, 'Monads, Effects and Transformations'. In: A. Gordon and A. Pitts (eds.): *Electronic Notes in Theoretical Computer Science*, Vol. 26. Elsevier Science Publishers.

Benton, N. and P. Wadler: 1996, 'Linear Logic, Monads and the Lambda Calculus'. In: *Proceedings, 11$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science*. New Brunswick, pp. 420–431.

Felleisen, M. and D. Friedman: 1986, 'Control operators, the SECD-machine, and the λ-calculus'. In: M. Wirsing (ed.): *Formal Description of Programming Concepts*. North-Holland.

Filinski, A.: 1996, 'Controlling Effects'. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Hatcliff, J.: 1994, 'The Structure of Continuation-Passing Styles'. Ph.D. thesis, Kansas State University.

Hatcliff, J. and O. Danvy: 1997, 'Thunks and the λ-calculus'. *Journal of Functional Programming* **7**(3), 303–319.

Howard, B. T.: 1996, 'Inductive, coinductive, and pointed types'. *ACM SIGPLAN Notices* **31**(6), 102–109.

Jeffrey, A.: 1999, 'A Fully Abstract Semantics for a Higher-order Functional Language with Nondeterministic Computation'. *Theoretical Comp. Sci.* **228**.

Krivine, J.-L.: 1985, 'Un interpréteur de λ-calcul'. Unpublished.

Laird, J.: 1998, 'A Semantic Analysis of Control'. Ph.D. thesis, University of Edinburgh.

Laurent, O.: 1999, 'Polarized proof-nets: proof-nets for LC (Extended Abstract)'. In: J.-Y. Girard (ed.): *Typed Lambda Calculi and Applications '99*, Vol. 1581 of *Lecture Notes in Computer Science*. pp. 213–227, Springer.

Levy, P. B., 'Adjunction Models For Call-By-Push-Value With Stacks'. to appear in Theory and Applications of Categories.

Levy, P. B.: 1996, 'λ-Calculus and Cartesian Closed Categories'. Essay for Part III of the Mathematical Tripos, Cambridge University, manuscript.

Levy, P. B.: 2002, 'Possible World Semantics for General Storage in Call-By-Value'. In: J. Bradfield (ed.): *Proc., 16th Annual Conference in Computer Science Logic, Edinburgh, 2002*, Vol. 2471 of *LNCS*. pp. 232–246, Springer.

Levy, P. B.: 2003, 'Adjunction Models For Call-By-Push-Value With Stacks'. In: *Proc., 9th Conference on Category Theory and Computer Science, Ottawa, 2002*, Vol. 69 of *Electronic Notes in Theoretical Computer Science*.

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

Marz, M.: 1998, 'A Fully Abstract Model for Sequential Computation'. Technical Report CSR-98-6, University of Birmingham, School of Computer Science.

Marz, M.: 2000, 'A Fully Abstract Model for Sequential Computation'. Ph.D. thesis, Technische Universität Darmstadt. published by Logos-Verlag, Berlin.

Moggi, E.: 1988, 'Computational Lambda-calculus and Monads'. LFCS Report ECS-LFCS-88-66, University of Edinburgh.

Moggi, E.: 1991, 'Notions of Computation and Monads'. *Inf. and Comp.* **93**.

O'Hearn, P. W.: 1993, 'Opaque Types in Algol-like Languages'. manuscript.

Pitts, A. M. and I. D. B. Stark: 1998, 'Operational Reasoning for Functions with Local State'. In: A. D. Gordon and A. M. Pitts (eds.): *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute. Cambridge University Press, pp. 227–273.

Plotkin, G. D.: 1976, 'Call-by-Name, Call-by-Value and the λ-Calculus'. *Theoretical Computer Science* **1**(1), 125–159.

Selinger, P.: 2001, 'Control Categories and Duality: On the Categorical Semantics of the $\lambda\mu$-Calculus'. *Mathematical Structures in Computer Science* **11**(2).

Streicher, T. and B. Reus: 1998, 'Classical logic, continuation semantics and abstract machines'. *Journal of Functional Programming* **8**(6).

Tait, W. W.: 1967, 'Intensional Interpretation of Functionals of Finite Type I'. *Journal of Symbolic Logic* **32**(2), 198–212.

Winskel, G.: 1993, *The Formal Semantics of Programming Languages.* Cambridge, Massachusetts: MIT Press.

Appendix

We give here the proof of termination (for CBPV without recursion) and that computations denoting $\bot$ diverge (for CBPV with recursion). Both of these are adaptations of standard arguments based on the method of (Tait, 1967).

Here is the proof of Prop. 1.

Proof. Determinism is trivial in every case. For termination, we use a Tait-style proof. Here it is for storage; the other proofs are similar. We define, by mutual induction over types, three families of subsets:

- for each value type A, a set $\mathsf{red}^{\mathsf{v}}_{A}$ of closed values of type A
- for each computation type $\underline{B}$, a set $\mathsf{red}^{\mathsf{t}}_{\underline{B}}$ of pairs s, T where $s \in S$ and T is a terminal computation of type $\underline{B}$
- for each computation type $\underline{B}$, a set $\mathsf{red}^{\mathsf{c}}_{\underline{B}}$ of pairs s, M where $s \in S$ and M is a closed computation of type $\underline{B}$

The definition of these subsets proceeds as follows:

$$\begin{array}{lll}
\mathtt{thunk}\ M \in \mathsf{red}^{\mathsf{v}}_{U\underline{B}} & \text{iff} & s, M \in \mathsf{red}^{\mathsf{c}}_{\underline{B}} \text{ for all } s \in S \\
(\hat{\imath}, V) \in \mathsf{red}^{\mathsf{v}}_{\sum_{i \in I} A_i} & \text{iff} & V \in \mathsf{red}^{\mathsf{v}}_{A_{\hat{\imath}}} \\
(V, V') \in \mathsf{red}^{\mathsf{v}}_{A \times A'} & \text{iff} & V \in \mathsf{red}^{\mathsf{v}}_{A} \text{ and } V' \in \mathsf{red}^{\mathsf{v}}_{A'} \\
 & & \\
s, \mathtt{return}\ V \in \mathsf{red}^{\mathsf{t}}_{FA} & \text{iff} & V \in \mathsf{red}^{\mathsf{v}}_{A} \\
s, \lambda\{\ldots, i.M_i, \ldots\} \in \mathsf{red}^{\mathsf{t}}_{\prod_{i \in I} \underline{B}_i} & \text{iff} & s, M_i \in \mathsf{red}^{\mathsf{c}}\underline{B}_i \text{ for all } i \in I \\
s, \lambda \mathtt{x}.M \in \mathsf{red}^{\mathsf{t}}_{A \to \underline{B}} & \text{iff} & s, M[V/\mathtt{x}] \in \mathsf{red}^{\mathsf{c}}_{\underline{B}} \text{ for all } V \in \mathsf{red}^{\mathsf{v}}_{A} \\
 & & \\
s, M \in \mathsf{red}^{\mathsf{c}}_{\underline{B}} & \text{iff} & s, M \Downarrow s', T \text{ for some } s', T \in \mathsf{red}^{\mathsf{t}}_{\underline{B}}
\end{array}$$

Notice that if T is terminal then $T \in \mathsf{red}^{\mathsf{c}}_{\underline{B}}$ iff $T \in \mathsf{red}^{\mathsf{t}}_{\underline{B}}$.

Finally we show that for any computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, if $W_i \in \mathsf{red}^{\mathsf{v}}_{A_i}$ for $i = 0, \ldots, n-1$ then $M[\overrightarrow{W_i/\mathtt{x}_i}] \in \mathsf{red}^{\mathsf{c}}_{\underline{B}}$; and similarly for any value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : A$. This is shown by mutual induction on M and V, and gives the required result.

Here is the proof of Prop. 2.

Proof. These are all proved by induction on $\Downarrow$, except the clause about divergence which requires a Tait-style proof. We define, by mutual induction over types, three families of relations:

- for each value type A, a relation $\leqslant^{\mathsf{v}}_{A}$ between $[\![A]\!]$ and closed values of type A such that, for each V, the set $\{a|a \leqslant^{\mathsf{v}}_{A} V\}$ is admissible and has the property[6] that if it contains a, a' where $\mathsf{return}\ a \leqslant \mathsf{return}\ a'$ then $a \leqslant a'$

- for each computation type $\underline{B}$ a relation $\leqslant^{\mathsf{t}}_{\underline{B}}$ between $[\![\underline{B}]\!]$ and terminal computations of type $\underline{B}$, such that, for each T, the set $\{a|a \leqslant^{\mathsf{t}}_{\underline{B}} T\}$ is admissible and contains $\bot$

- for each computation type $\underline{B}$ a relation $\leqslant^{\mathsf{c}}_{\underline{B}}$ between $[\![\underline{B}]\!]$ and closed computations of type $\underline{B}$, such that, for each M, the set $\{a|a \leqslant^{\mathsf{c}}_{\underline{B}} M\}$ is admissible and contains $\bot$.

The definition of these relations proceeds as follows:

$$
\begin{array}{lll}
a \leqslant^{\mathsf{v}}_{U\underline{B}} \mathtt{thunk}\ M & \text{iff} & \mathsf{force}\ a \leqslant^{\mathsf{c}}_{\underline{B}} M \\
a \leqslant^{\mathsf{v}}_{\sum_{i\in I} A_i} (\hat{\imath}, V) & \text{iff} & a = (\hat{\imath}, b) \text{ for some } b \leqslant^{\mathsf{v}}_{A_{\hat{\imath}}} V \\
a \leqslant^{\mathsf{v}}_{A\times A'} (V, V') & \text{iff} & a = (b, b') \text{ for some } b \leqslant^{\mathsf{v}}_{A} V \text{ and } b' \leqslant^{\mathsf{v}}_{A'} V' \\
 & & \\
b \leqslant^{\mathsf{t}}_{FA} \mathtt{return}\ V & \text{iff} & b = \bot \text{ or } b = \mathsf{return}\ a \text{ for some } a \leqslant^{\mathsf{v}}_{A} V \\
f \leqslant^{\mathsf{t}}_{\prod_{i\in I}\underline{B}_i} \lambda\{\ldots, i.M_i, \ldots\} & \text{iff} & \hat{\imath} \in I \text{ implies } \hat{\imath}`f \leqslant^{\mathsf{c}}_{\underline{B}_i} M_{\hat{\imath}} \\
f \leqslant^{\mathsf{t}}_{A\to\underline{B}} \lambda\mathtt{x}.M & \text{iff} & a \leqslant^{\mathsf{v}}_{A} V \text{ implies } a`f \leqslant^{\mathsf{c}}_{\underline{B}} M[V/\mathtt{x}] \\
 & & \\
b \leqslant^{\mathsf{c}}_{\underline{B}} M & \text{iff} & b = \bot \text{ or, for some } T,\ M \Downarrow T \text{ and } b \leqslant^{\mathsf{t}}_{\underline{B}} T
\end{array}
$$

Notice that, if T is terminal, then $b \leqslant^{\mathsf{c}}_{\underline{B}} T$ iff $b \leqslant^{\mathsf{t}}_{\underline{B}} T$. Next, we show that for any computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, if $a_i \leqslant^{\mathsf{v}}_{A_i} W_i$ for $i = 0, \ldots, n-1$ then $[\![M]\!]\overrightarrow{\mathtt{x}_i \mapsto a_i} \leqslant^{\mathsf{c}}_{\underline{B}} M[\overrightarrow{W_i/\mathtt{x}_i}]$; and similarly for any value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : A$. This is shown by mutual induction on M and V, and gives the required result.

[6] This clause, which is used in proving the admissibility property for FA, is redundant in the Scott model because $\mathsf{return}\ a \leqslant \mathsf{return}\ a'$ always implies $a \leqslant a'$. However, we include it here so that the proof generalizes to other models where the stronger result is not valid.