Higher-Order Symb Comput (2006) 19:377–414
DOI 10.1007/s10990-006-0480-6

Call-by-push-value: Decomposing call-by-value and call-by-name

Paul Blain Levy

Abstract We present the call-by-push-value (CBPV) calculus, 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's 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.

Keywords Call-by-push-value · Computational effect · Monad · Lambda-calculus · Call-by-value · Call-by-name

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

P. B. Levy (✉)
University of Birmingham, UK
e-mail: pbl@cs.bham.ac.uk

– the subsuming language, like CBV and CBN, is equipped with operational semantics, cpo semantics, monad semantics, storage semantics in the manner of [27] and continuation semantics in the manner of [33]
– 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 [7], a form of the monadic metalanguage [26]. 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 Section 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. Section 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 [11], 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 [5] preserve cpo semantics, but not the others.
– Translations into SFPL [24] preserve cpo semantics and operational semantics, but not the others.
– Translations into continuation languages [29], or their polarized counterpart LLP [15], preserve continuation semantics, including unbracketed game semantics [14], and (certain) operational semantics, but not the others. Likewise the related work of [32].

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 [20] for more information.

1.3 Structure of paper

Before looking at CBPV, we develop some themes in monad semantics of CBN (Section 2.1) and CBV (Section 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 Section 6.2, which presents a *behaviour semantics* for storage—actually the simpler CBPV version of the CBN semantics in [27]. 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.[1]

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 we also need 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's SFPL (except for recursive types, which we do not treat in this paper).

1.4 Note

Since the original presentation in [17], some changes have been made to the CBPV syntax (including recursion).

$$\begin{aligned}
\texttt{produce}\, V &\rightsquigarrow \texttt{return}\, V \\
\langle \ldots, i.M_i, \ldots_{i\in I}\rangle &\rightsquigarrow \lambda\{i.M_i\}_{i\in I} \\
\texttt{let x be}\, V.\, M &\rightsquigarrow \texttt{let}\, V\, \texttt{be x}.\, M \\
(\hat{\imath}, V) &\rightsquigarrow \langle \hat{\imath}, V\rangle \\
(V, V') &\rightsquigarrow \langle V, V'\rangle \\
\mu\texttt{x}.\, M &\rightsquigarrow \texttt{rec x}.\, M
\end{aligned}$$

2 Monads

2.1 Algebra semantics for call-by-name

There are two theories for typed CBN:

– the "lazy" theory [10, 28], where convergence is observed at every type
– the "PCF-style" theory [30], where convergence is observed at ground type only.

[1] In fact the latter can be deduced from the former, as explained at the end of Section 9.

The terms $\lambda \mathtt{x}_A.\mathtt{diverge}_B$ and $\mathtt{diverge}_{A\to B}$ are observationally equivalent in the PCF-style theory, but not in the lazy theory. (The lazy theory has also been studied in the untyped setting [1, 10, 28, 29].)

Moggi's seminal paper [26] provided translations from CBV and lazy 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 translation from lazy CBN is shown in [9] to be the composite

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

Here the first factor is the *thunking transform* of [10].

A semantics of PCF-style CBN in $(\mathcal{C}, T)$ is given in [3, 7]. 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—which, for reasons explained below, we call *carrier* semantics—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

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

in the sense that the denotation of the LHS is defined to be that of the RHS. Furthermore, in order to prove the computational adequacy of such a semantics, one first has to prove[2] that (1) is an observational equivalence.

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 $X \in \mathtt{ob}\,\mathcal{C}$ and $TX \xrightarrow{\theta} X$ is a morphism such that

$$\begin{array}{ccccc} X & \xrightarrow{\eta X} & TX & \xrightarrow{\mu X} & T^2X \\ & \searrow^{\mathrm{id}} & \downarrow{\scriptstyle \theta} & & \downarrow{\scriptstyle T\theta} \\ & & X & \xrightarrow[\theta]{} & TX \end{array} \tag{2}$$

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.

[2] An alternative method is given in Remark 1 below.

– 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. This corresponds to a monoid action, written $**$, of $\mathcal{A}^*$ on X.

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

– The *free* T-algebra on a $\mathcal{C}$-object A has carrier TA and structure map μA.
– For a family of T-algebras $\{(X_i, \theta_i)\}_{i \in I}$, suppose the object family $\{X_i\}_{i \in I}$ has a product, with vertex V and projection $V \xrightarrow{\pi_i} X_i$ for each $i \in I$. Then the *product algebra* $\prod_{i \in I}(X_i, \theta_i)$ has carrier V and structure the unique $TV \xrightarrow{\phi} V$ such that

$$\begin{array}{ccc} TV & \xrightarrow{T\pi_i} & TX_i \\ {\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\theta_i} \\ V & \xrightarrow[\pi_i]{} & X_i \end{array} \qquad \text{commutes for each } i \in I.$$

– For a $\mathcal{C}$-object A and T-algebra (X, θ), suppose there is an exponential from A to X, with vertex V and evaluation $A \times V \xrightarrow{\mathsf{ev}} X$. Then the *exponential algebra* $A \to (X, \theta)$ has carrier V and structure the unique $TV \xrightarrow{\phi} V$ such that

$$\begin{array}{ccccc} A \times TV & \xrightarrow{t_{A,V}} & T(A \times V) & \xrightarrow{T\mathsf{ev}} & TX \\ {\scriptstyle A\times\phi}\downarrow & & & & \downarrow{\scriptstyle\theta} \\ A \times V & & \xrightarrow[\mathsf{ev}]{} & & X \end{array} \qquad \text{commutes.}$$

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 algebra 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 $U^T[\![B]\!]$.

Clearly,

– every type's carrier denotation is the carrier of its algebra denotation (hence the name "carrier semantics")
– the two sides of (1) have the same denotation in algebra semantics (but not by definition, unlike in carrier semantics)
– hence each term has the same denotation in algebra and carrier semantics.

But in algebra semantics, the interpretation of conditionals is compositional.

Remark 1. The computational adequacy of carrier semantics can be deduced from that of algebra semantics, since the denotations of terms are the same.

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

$$A ::= \quad \texttt{bool} \mid (A_0, \ldots, A_{n-1}) \to A$$

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 [10,25], 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 interpret each type by a set:

$$[\![\texttt{bool}]\!] = 1 + 1$$

$$[\![(A_0, \ldots, A_{n-1}) \to B]\!] = (S \times [\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]) \to (S \times [\![B]\!])$$

Then we interpret each term $\Gamma \vdash M : B$ by a function $S \times [\![\Gamma]\!] \xrightarrow{[\![M]\!]} S \times [\![B]\!]$. Next, we interpret each value $\Gamma \vdash V : B$ by a function $[\![\Gamma]\!] \xrightarrow{[\![V]\!]^{\mathsf{val}}} [\![B]\!]$ such that $[\![V]\!](s, \rho) = (s, [\![V]\!]^{\mathsf{val}}\rho)$ for each $s \in S$ and $\rho \in [\![\Gamma]\!]$.

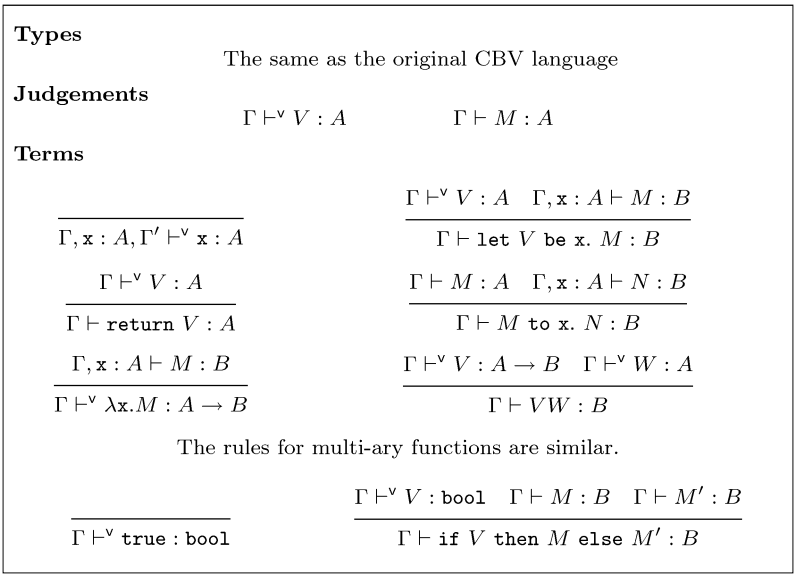

Fig. 1 Syntax of fine-grain CBV, for boolean and multi-ary function types

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 three 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 $\Gamma \vdash^{\mathrm{v}} V : A$ for values, and to make explicit the coercion of values into effectful terms. Doing this gives something like the *fine-grain CBV* language[3] shown in Fig. 1. This greatly simplifies the semantics of CBV constructs, e.g. application.

The second step, a rather minor one, is to allow *values* to be formed using `let` and `if`, by adding the typing rules

$$\frac{\Gamma \vdash^{\mathrm{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathrm{v}} W : B}{\Gamma \vdash^{\mathrm{v}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ W : B} \qquad \frac{\Gamma \vdash^{\mathrm{v}} V : \mathtt{bool} \quad \Gamma \vdash^{\mathrm{v}} W : B \quad \Gamma \vdash^{\mathrm{v}} W' : B}{\Gamma \vdash^{\mathrm{v}} \mathtt{if}\ V\ \mathtt{then}\ W\ \mathtt{else}\ W' : B}$$

These so-called *complex* values greatly complicate the operational semantics, because they need to be evaluated. However, they are very natural from a denotational and categorical viewpoint. We discuss complex values in detail in Section 8.2 (not in the CBV setting, but everything we say applies equally to CBV).

[3] The language MIL-lite in [4], 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.

The third step is to observe that

$$[\![() \to B]\!] = S \to (S \times [\![B]\!]) \tag{3}$$

$$[\![\lambda().M]\!]^{\mathsf{val}}\rho = \lambda s.([\![M]\!](s, \rho)) \tag{4}$$

$$[\![V()]\!](s, \rho) = ([\![V]\!]^{\mathsf{val}}\rho)s \tag{5}$$

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 this third step is problematic. Firstly, because it erases the conceptually significant difference between a CBV term and its thunk. Secondly, 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 [18], 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, but a full treatment can be found in [20].

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

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 Section 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

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

where I is any finite[4] set. The elements of I are called *tags*, and we write them starting with #, to avoid confusion with identifiers. For example,

$$\textstyle\sum\{\#\mathsf{jan}.A, \#\mathsf{feb}.B, \#\mathsf{mar}.C\}$$

is a value type if A, B, C are value types.

[4] 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 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.

The type 1 is entirely analogous to $\times$, so we generally omit typing rules, etc., for it.

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 distinct identifiers with associated value types. We often omit the identifiers and write just $A_0, \ldots, A_{n-1}$. In an algebra building structure this denotes the $\mathcal{C}$-object $[\![\Gamma]\!] = [\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$.

We write $\Gamma \vdash^{\mathtt{v}} V : A$ to mean that V is a value of type A, and we write $\Gamma \vdash^{\mathtt{c}} M : \underline{B}$ 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 Section 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 idiom >>= λ. Thus M `to` x. N (unlike in Haskell, N can have any computation type) 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.

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

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

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

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

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

$$\frac{\Gamma \vdash^{\mathtt{c}} M_i : \underline{B}_i\ (\forall i \in I)}{\Gamma \vdash^{\mathtt{c}} \lambda\{i.M_i\}_{i\in I} : \prod_{i\in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathtt{c}} M : \prod_{i\in I} \underline{B}_i}{\Gamma \vdash^{\mathtt{c}} \hat{\imath}\text{‘}M : \underline{B}_{\hat{\imath}}}\ \hat{\imath} \in I$$

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

Fig. 2 Terms of CBPV

We write

$A + B$	for	$\sum\{\#\mathsf{l}.A, \#\mathsf{r}.B\}$
0	for	$\sum\{\}$
$\underline{A} \sqcap \underline{B}$	for	$\prod\{\#\mathsf{l}.A, \#\mathsf{r}.B\}$
1_Π	for	$\prod\{\}$

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

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

– 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

$$[\![\Gamma]\!] \xrightarrow{(\mathsf{id}, [\![M]\!])} [\![\Gamma]\!] \times T[\![A]\!] \xrightarrow{t_{[\![\Gamma]\!],[\![A]\!]}} T([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{T[\![N]\!]} TY \xrightarrow{\phi} Y$$

– `thunk` and `force` are invisible.

In Section 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.

The denotational semantics of divergence in the cppo model is

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

Divergence

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

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

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{print}\ c.\ M : \underline{B}}\ c \in \mathcal{A}$$

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

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{cell} := s.\ M : \underline{B}}\ s \in S \qquad \frac{\Gamma \vdash^{\mathsf{c}} M_s : \underline{B}\ \ (\forall s \in S)}{\Gamma \vdash^{\mathsf{c}} \mathtt{read\text{-}cell\text{-}as}\ \{s.M_s\}_{s \in S} : \underline{B}}$$

Fig. 3 Adding divergence, printing, storage

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

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

We discuss denotational semantics of storage in Section 6.1.

4 Big-step semantics

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

$$T ::= \quad \texttt{return}\ V \mid \lambda\{i.M_i\}_{i\in I} \mid \lambda\mathtt{x}.M$$

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 Figs. 4 and 5.

Each big-step rule has the form

$$\frac{M_0 \Downarrow T_0 \quad \cdots \quad M_{r-1} \Downarrow T_{r-1}}{M \Downarrow T} \tag{6}$$

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

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

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

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

$$\frac{M_{\hat{\imath}}[V/\mathtt{x}] \Downarrow T}{\texttt{pm}\ \langle \hat{\imath}, V\rangle\ \texttt{as}\ \{\langle i, \mathtt{x}\rangle.M_i\}_{i\in I} \Downarrow T}$$

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

$$\frac{}{\lambda\{i.M_i\}_{i\in I} \Downarrow \lambda\{i.M_i\}_{i\in I}} \qquad \frac{M \Downarrow \lambda\{i.N_i\}_{i\in I} \quad N_{\hat{\imath}} \Downarrow T}{\hat{\imath}\text{`}M \Downarrow T}$$

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

Fig. 4 Big-step semantics for CBPV

Divergence We add the rules

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

and we say that M *diverges* 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 (6) by

$$\frac{M_0 \Downarrow m_0, T_0 \quad \cdots \quad M_{r-1} \Downarrow m_{r-1}, T_{r-1}}{M \Downarrow m_0 + m_1 + \cdots + m_{r-1}, T}$$

where $+$ means concatenation of strings, 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 (6) 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

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

Fig. 5 Big-step semantics for divergence, printing and storage

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.

Definition 2. For any computation $\vdash^{\mathsf{c}} M : \sum_{i \in I} 1$ (such a computation is said to be *ground*), its *operation* $[M]$ is

no effects an element of I. If $M \Downarrow \texttt{return } \langle i, \langle\rangle\rangle$ then $[M] = i$
divergence an element of $I_\perp$. If $M \Downarrow \texttt{return } \langle i, \langle\rangle\rangle$ then $[M] = \mathsf{up}\, i$, and if M diverges then $[M] = \perp$
printing an element of $\mathcal{A}^* \times I$. If $M \Downarrow m, \texttt{return } \langle i, \langle\rangle\rangle$ then $[M] = (m, i)$
storage an element of $S \to (S \times I)$: for each $s \in S$, if $s, M \Downarrow s', \texttt{return } \langle i, \langle\rangle\rangle$ then $[M]s = (s', i)$.

In each case, we define *observational equivalence* $\simeq$ to be the largest congruence on terms such that if $M \simeq M'$ then $[M] = [M']$. More explicitly, for two computations $\Gamma \vdash^{\mathsf{c}} M, M' :$

$\underline{B}$, we say $M \simeq M'$ when $[\mathcal{C}[M]] = [\mathcal{C}[M']]$ for any ground-computation context $\mathcal{C}$ with hole $\Gamma \vdash^{\mathsf{c}} [\cdot] : \underline{B}$.

In the case of divergence, we define *observational inequality* $\lesssim$ to be the largest precongruence on terms such that if $M \lesssim M'$ then $[M] \leqslant [M']$. More explicitly, for two computations $\Gamma \vdash^{\mathsf{c}} M, M' : \underline{B}$, we say $M \lesssim M'$ when $[\mathcal{C}[M]] \leqslant [\mathcal{C}[M']]$ for any ground-computation context $\mathcal{C}$ with hole $\Gamma \vdash^{\mathsf{c}} [\cdot] : \underline{B}$.

In Section 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. We write ϵ for the empty environment.*

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

The proof is standard, and in the Appendix.

Corollary 1. *Both for divergence and for printing, we have $[\![N]\!]\epsilon = [N]$ for every ground computation N. Hence for any terms M, M' (not necessarily closed), if $[\![M]\!] = [\![M']\!]$ then $M \simeq M'$, by compositionality of $[\![-]\!]$. In the case of divergence, $[\![M]\!] \leqslant [\![M']\!]$ implies $M \lesssim M'$.*

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 [6] in a CBV setting, and there are many similar formulations [13, 31, 33].

At any point in time, the machine has configuration M, K when M is the computation we are evaluating and K is a stack. Here is a "raw" (i.e. type free) grammar of stacks:

$$K ::= \quad \mathtt{nil} \mid \mathtt{to}\ \mathtt{x}.\ N :: K \mid V :: K \mid \hat{\imath} :: K$$

For a typed grammar, see [21].

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

Initial Configuration

M	$\mathtt{nil}$

Transitions

$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M$	$K \rightsquigarrow$
$M[V/\mathtt{x}]$	K
$M\ \mathtt{to}\ \mathtt{x}.\ N$	$K \rightsquigarrow$
M	$\mathtt{to}\ \mathtt{x}.\ N :: K$
$\mathtt{return}\ V$	$\mathtt{to}\ \mathtt{x}.\ N :: K \rightsquigarrow$
$N[V/\mathtt{x}]$	K
$\mathtt{force}\ \mathtt{thunk}\ M$	$K \rightsquigarrow$
M	K
$\mathtt{pm}\ \langle \hat{\imath}, V\rangle\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.M_i\}_{i\in I}$	$K \rightsquigarrow$
$M_{\hat{\imath}}[V/\mathtt{x}]$	K
$\mathtt{pm}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.M$	$K \rightsquigarrow$
$M[V/\mathtt{x}, V'/\mathtt{y}]$	K
$\hat{\imath}`M$	$K \rightsquigarrow$
M	$\hat{\imath} :: K$
$\lambda\{i.M_i\}_{i\in I}$	$\hat{\imath} :: K \rightsquigarrow$
$M_{\hat{\imath}}$	K
$V`M$	$K \rightsquigarrow$
M	$V :: K$
$\lambda\mathtt{x}.M$	$V :: K \rightsquigarrow$
$M[V/\mathtt{x}]$	K

Terminal Configurations

$\mathtt{return}\ V$	$\mathtt{nil}$
$\lambda\{i.M_i\}_{i\in I}$	$\mathtt{nil}$
$\lambda\mathtt{x}.M$	$\mathtt{nil}$

Fig. 6 CK-machine for CBPV

The machine is shown in Fig. 6. To evaluate a closed computation M, we start with the configuration $M, \mathtt{nil}$ and follow the transitions 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 [20] by standard techniques. For each of our effects, it is straightforward to adapt the CK-machine and obtain a variant of Proposition 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 λx can be read as an instruction “pop 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.
        return 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 $\langle i, V\rangle$, where $i \in I$ and V is a value of type A_i.
- A value of type $A \times A'$ is a pair $\langle V, V'\rangle$, 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 $\langle\,\rangle$.

- 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 Section 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 $s, M \Downarrow s', \mathtt{return}\ V$, then $(\llbracket M \rrbracket \epsilon)s = (s', \llbracket V \rrbracket \epsilon)$.*

This is not straightforward to prove. One 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 Section 9). We now turn to this other denotational model, called *behaviour semantics*, and introduced for CBN in [27]. 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 $\llbracket \Gamma \rrbracket$ to $\llbracket A \rrbracket$.

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 $S \times \llbracket \Gamma \rrbracket$ to $\llbracket \underline{B} \rrbracket$, because, in a given store $s \in S$ and environment $\rho \in \llbracket \Gamma \rrbracket$, 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 $S \times \llbracket A \rrbracket$.
- 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 $\llbracket A \rrbracket \to \llbracket \underline{B} \rrbracket$.
- 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 $\prod_{i \in I} \llbracket \underline{B}_i \rrbracket$.
- 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 $S \to \llbracket \underline{B} \rrbracket$.

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 $\llbracket M \rrbracket(s, \epsilon) = \llbracket T \rrbracket(s', \epsilon)$.*

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

Corollary 2. *$[\![N]\!](s, \epsilon) = [N]s$ for every ground computation N and $s \in S$. Hence, for any terms M, M', if $[\![M]\!] = [\![M']\!]$ then $M \simeq M'$.*

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

$$A \to_{\mathbf{CBV}} B = U(A \to FB) \tag{7}$$

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

$$\mathtt{bool}_{\mathbf{CBN}} = F(1 + 1)$$
$$\underline{A} +_{\mathbf{CBN}} \underline{A'} = F(U\underline{A} + U\underline{A'})$$

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 Section 7.4.

The types we treat for CBV and CBN are

$$A ::= \quad \mathtt{bool} \mid A \to A \mid A + A \tag{8}$$

deferring the treatment of other connectives to Section 7.5. For the connectives in (8), the term syntax and the big-step semantics ($\Downarrow_{\mathsf{CBN}}$ and $\Downarrow_{\mathsf{CBV}}$) are standard e.g. [35]. For recursion, there are several possible formulations, one of which is shown in Fig. 7. Observational equivalence ($\simeq_{\mathsf{CBN}}$ and $\simeq_{\mathsf{CBV}}$) and inequality ($\lesssim_{\mathsf{CBN}}$ and $\lesssim_{\mathsf{CBV}}$) are defined as in Definition 2, defining ground terms to be closed terms of type `bool`.

7.2 CBN to CBPV

Each CBN term $A_0, \ldots, A_{n-1} \vdash M : B$ is translated into a CBPV computation $UA_0^{\mathsf{n}}, \ldots, UA_{n-1}^{\mathsf{n}} \vdash^{\mathsf{c}} M^{\mathsf{n}} : B^{\mathsf{n}}$. 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.` λ`y.x`.

To achieve preservation of substitution, and preservation and reflection of big-step semantics (we require reflection so that our account extends to a nondeterministic setting), we work with a *relation* $\mapsto^{\mathsf{n}}$ from CBN terms to CBPV computations. This is defined inductively;

CBN

$$\frac{\Gamma, \mathtt{x} : A \vdash M : A}{\Gamma \vdash \mathtt{rec\ x.}\ M : A} \qquad \frac{M[\mathtt{rec\ x.}\ M/\mathtt{x}] \Downarrow_{\mathsf{CBN}} T}{\mathtt{rec\ x.}\ M \Downarrow_{\mathsf{CBN}} T}$$

CBV

$$\frac{\Gamma, \mathtt{x} : A, \mathtt{f} : A \to B \vdash M : B}{\Gamma \vdash \mathtt{rec\ f}\ \lambda \mathtt{x}.M : A \to B} \qquad \frac{}{\mathtt{rec\ f}\ \lambda \mathtt{x}.M \Downarrow_{\mathsf{CBV}} \mathtt{rec\ f}\ \lambda \mathtt{x}.M}$$

$$\frac{M \Downarrow_{\mathsf{CBV}} \mathtt{rec\ f}\ \lambda \mathtt{x}.P \quad N \Downarrow_{\mathsf{CBV}} V \quad P[\mathtt{rec\ f}\ \lambda \mathtt{x}.P/\mathtt{f}, V/\mathtt{x}] \Downarrow_{\mathsf{CBV}} W}{MN \Downarrow_{\mathsf{CBV}} W}$$

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

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

$A_0, \ldots, A_{n-1} \vdash M : C$	$UA^{\mathsf{n}}_0, \ldots, UA^{\mathsf{n}}_{n-1} \vdash^{\mathsf{c}} M^{\mathsf{n}} : C^{\mathsf{n}}$
$\mathtt{x}$	$\mathtt{force\ x}$
$\mathtt{let}\ M\ \mathtt{be\ x.}\ N$	$\mathtt{let\ thunk}\ M^{\mathsf{n}}\ \mathtt{be\ x.}\ N^{\mathsf{n}}$
$\mathtt{true}$	$\mathtt{return}\ \langle \#\mathsf{l}, \langle\rangle \rangle$
$\mathtt{false}$	$\mathtt{return}\ \langle \#\mathsf{r}, \langle\rangle \rangle$
$\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N'$	$M^{\mathsf{n}}\ \mathtt{to\ z.\ pm\ z\ as}\ \{\langle \#\mathsf{l}, \mathtt{u}\rangle.N^{\mathsf{n}}, \langle \#\mathsf{r}, \mathtt{u}\rangle.N'^{\mathsf{n}}\}$
$\mathtt{inl}\ M$	$\mathtt{return}\ \langle \#\mathsf{l}, \mathtt{thunk}\ M^{\mathsf{n}} \rangle$
$\mathtt{inr}\ M$	$\mathtt{return}\ \langle \#\mathsf{r}, \mathtt{thunk}\ M^{\mathsf{n}} \rangle$
$\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N, \mathtt{inr\ x}.N'\}$	$M^{\mathsf{n}}\ \mathtt{to\ z.\ pm\ z\ as}\ \{\langle \#\mathsf{l}, \mathtt{x}\rangle.N^{\mathsf{n}}, \langle \#\mathsf{r}, \mathtt{x}\rangle.N'^{\mathsf{n}}\}$
$\lambda \mathtt{x}.M$	$\lambda \mathtt{x}.M^{\mathsf{n}}$
MN	$(\mathtt{thunk}\ N^{\mathsf{n}})\text{`}M^{\mathsf{n}}$
$\mathtt{rec\ x.}\ M$	$\mathtt{rec\ x.}\ M^{\mathsf{n}}$
$\mathtt{print}\ c.\ M$	$\mathtt{print}\ c.\ M^{\mathsf{n}}$

Fig. 8 Translating CBN to CBPV

there is one rule for each line of Fig. 8 e.g.

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

together with an additional rule

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

Now substitution is preserved.

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

Proposition 7. *The relation from CBN to CBPV terms is a bisimulation:*

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

Proof: For (1), induct, primarily on $M \Downarrow_{\mathsf{CBN}} T$ and secondarily on $M \mapsto^{\mathsf{n}} M'$. For (1), induct on $M' \Downarrow T'$. □

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 Section 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.

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

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

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

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

$A_0,\ldots,A_{n-1} \vdash M : C$	$A_0^{\mathsf{v}},\ldots,A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{c}} M^{\mathsf{v}} : FC^{\mathsf{v}}$
x	return x
let M be x. N	M^{v} to x. N^{v}
true	return $\langle \#\mathsf{l}, \langle\rangle \rangle$
false	return $\langle \#\mathsf{r}, \langle\rangle \rangle$
if M then N else N'	M^{v} to z. pm z as $\{\langle \#\mathsf{l}, \mathsf{u}\rangle.N^{\mathsf{v}}, \langle \#\mathsf{r}, \mathsf{u}\rangle.N'^{\mathsf{v}}\}$
inl M	M^{v} to z. return $\langle \#\mathsf{l}, \mathsf{z}\rangle$
inr M	M^{v} to z. return $\langle \#\mathsf{r}, \mathsf{z}\rangle$
pm M as $\{\mathtt{inl\ x}.N, \mathtt{inr\ x}.N'\}$	M^{v} to z. pm z as $\{\langle \#\mathsf{l}, \mathsf{x}\rangle.N^{\mathsf{v}}, \langle \#\mathsf{r}, \mathsf{x}\rangle.N'^{\mathsf{v}}\}$
$\lambda \mathtt{x}.M$	return thunk $\lambda \mathtt{x}.M^{\mathsf{v}}$
MN	M^{v} to f. N^{v} to x. x‘(force f)
print c. M	print c. M^{v}
rec f $\lambda \mathtt{x}.M$	return thunk rec f. $\lambda \mathtt{x}.M^{\mathsf{v}}$

$A_0,\ldots,A_{n-1} \vdash V : C$	$A_0^{\mathsf{v}},\ldots,A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{v}} V^{\mathsf{val}} : C^{\mathsf{v}}$
x	x
true	$\langle \#\mathsf{l}, \langle\rangle \rangle$
false	$\langle \#\mathsf{r}, \langle\rangle \rangle$
inl V	$\langle \#\mathsf{l}, V^{\mathsf{val}} \rangle$
inr V	$\langle \#\mathsf{r}, V^{\mathsf{val}} \rangle$
$\lambda \mathtt{x}.M$	thunk $\lambda \mathtt{x}.M^{\mathsf{v}}$
rec f $\lambda \mathtt{x}.M$	thunk rec f. $\lambda \mathtt{x}.M^{\mathsf{v}}$

Fig. 9 Translation of CBV types, terms and values

To these rules we add the following

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

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

Proposition 8. *The relations satisfy the following basic properties.*

1. *If* $V \mapsto^{\mathsf{val}} V'$ *then* $V \mapsto^{\mathsf{v}} \texttt{return}\ V'$.
2. *If* $M \mapsto^{\mathsf{v}} M'$ *and* $V \mapsto^{\mathsf{val}} V'$ *then* $M[V/\mathtt{x}] \mapsto^{\mathsf{v}} M'[V'/\mathtt{x}]$.
3. *If* $W \mapsto^{\mathsf{val}} W'$ *and* $V \mapsto^{\mathsf{val}} V'$ *then* $W[V/\mathtt{x}] \mapsto^{\mathsf{val}} W'[V'/\mathtt{x}]$.

The relation from CBV to CBPV terms is a bisimulation, in the following sense.

Proposition 9. *Suppose* $M \mapsto^{\mathsf{v}} M'$.

1. *If* $M \Downarrow_{\mathsf{CBV}} V$, *then there exists* V' *such that* $M' \Downarrow \texttt{return}\ V'$ *and* $V \mapsto^{\mathsf{val}} V'$.
2. *If* $M' \Downarrow \texttt{return}\ V'$ *there exists* V *such that* $M \Downarrow_{\mathsf{CBV}} V$ *and* $V \mapsto^{\mathsf{val}} V'$.

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

To prove this, we introduce the following.

Definition 3. We define two classes of *safe* terms.

1. In CBV, the following terms are safe:

$$\begin{aligned} S ::= \quad & \mathtt{x} \mid \texttt{let}\ S\ \texttt{be x.}\ S \mid \texttt{inl}\ S \mid \texttt{inr}\ S \\ & \mid \texttt{true} \mid \texttt{false} \mid \texttt{if}\ S\ \texttt{then}\ S\ \texttt{else}\ S \\ & \mid \texttt{pm}\ S\ \texttt{as}\ \{\texttt{inl x.}S, \texttt{inr x.}S\} \mid \lambda\mathtt{x}.M \end{aligned}$$

2. In CBPV the following terms (all computations of F type) are safe:

$$\begin{aligned} S ::= \quad & \texttt{return}\ V \mid \texttt{let}\ V\ \texttt{be x.}\ S \mid S\ \texttt{to x.}\ S \\ & \mid \texttt{pm}\ V\ \texttt{as}\ \{\langle i, \mathtt{x}\rangle.S_i\}_{i\in I} \mid \texttt{pm}\ V\ \texttt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.S \end{aligned}$$

Lemma 1. *Suppose* $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1} \vdash M : B$ *and* $M \mapsto^{\mathsf{v}} M'$. *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* $M[\overrightarrow{U_i/\mathtt{x}_i}] \Downarrow_{\mathsf{CBV}} V$, *then, for some* V', *we have* $M'[\overrightarrow{U_i'/\mathtt{x}_i}] \Downarrow \texttt{return}\ V'$ *and* $V \mapsto^{\mathsf{val}} V'$.
   - *If* $M'[\overrightarrow{U_i'/\mathtt{x}_i}] \Downarrow \texttt{return}\ V'$, *then, for some* V, *we have* $M[\overrightarrow{U_i/\mathtt{x}_i}] \Downarrow_{\mathsf{CBV}} V$ *and* $V \mapsto^{\mathsf{val}} V'$.

We prove Lemma 1 by induction on $M \mapsto^{\mathsf{v}} M'$.

Because of Lemma 1, we have the case $M' = \texttt{return}\ V'$ of Proposition 9. Using this fact, we prove Proposition 9(1) by induction on $M \Downarrow_{\mathsf{CBV}} V$, and (2) by induction on $M' \Downarrow \texttt{return}\ V'$.

7.4 Preservation of denotational semantics

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

Proposition 10. *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 Section* 2.

In particular, the monad semantics[5] of $U(A \to FB)$ is an exponential from $[\![A]\!]$ to $T[\![B]\!]$.

Proposition 11. *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* [27].

In particular, we have

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

7.5 Full abstraction

A frequently asked question is whether the translations from CBV and CBN to CBPV are fully abstract. This is not one but many questions: its meaning depends not only on the set of effects available, but also on the set of connectives provided in the source language. As illustration of this latter point, recall that the CBN language we have considered so far contains binary sum, but not ternary sum, which is not isomorphic to $(A + B) + C$ in CBN. Yet a CBN ternary sum can be represented in CBPV. So the question of whether the translation to CBPV is fully abstract incorporates the question of whether adding ternary sum to CBN is fully abstract; and this is non-trivial.

To avoid this problem, we want to suppose the CBV/CBN source languages to provide a "complete" range of connectives. It is argued in [22] that the canonical way of doing this is to provide two general connectives: $\boxed{\sum}$ (a kind of sum of products), and $\boxed{\prod}$ (a kind of products of multi-ary function types).

An example of $\boxed{\sum}$ is the following. If A, B, C are types, then $\boxed{\sum}\{\#\mathsf{l}.A, B; \#\mathsf{r}.C\}$ is the type of

– tuples $\langle \#\mathsf{l}, M, M' \rangle$, where M has type A and M' has type B, and
– tuples $\langle \#\mathsf{r}, M \rangle$, where M has type C.

[5] This in fact is true for *any* semantics of CBPV—this follows from Proposition 17 below.

This type has two introduction rules

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M' : B}{\Gamma \vdash \langle \#\mathsf{l}, M, M' \rangle : \boxed{\textstyle\sum}\{\#\mathsf{l}.A, B; \#\mathsf{r}.C\}} \qquad \frac{\Gamma \vdash M : C}{\Gamma \vdash \langle \#\mathsf{r}, M \rangle : \boxed{\textstyle\sum}\{\#\mathsf{l}.A, B; \#\mathsf{r}.C\}}$$

and one elimination rule that pattern-matches a tuple

$$\frac{\Gamma \vdash M : \boxed{\textstyle\sum}\{\#\mathsf{l}.A, B; \#\mathsf{r}.C\} \quad \Gamma, \mathtt{x} : A, \mathtt{y} : B \vdash N : D \quad \Gamma, \mathtt{x} : C \vdash N' : D}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\langle \#\mathsf{l}, \mathtt{x}, \mathtt{y} \rangle.\ N, \langle \#\mathsf{r}, \mathtt{x} \rangle.\ N'\} : D}$$

An example of $\boxed{\prod}$ is the following. If A,B,C,D,E are types, then $\boxed{\prod}\{\#\mathsf{l}.A,\ B \vdash C; \#\mathsf{r}.D \vdash E\}$ is the type of functions that

- map arguments $(\#\mathsf{l}, M, M')$, where M has type A and M' has type B, to something of type C, and
- map arguments $(\#\mathsf{r}, M)$, where M has type D, to something of type E.

This type has one introduction rule, a λ-abstraction that must provide *two* bodies:

$$\frac{\Gamma, \mathtt{x} : A, \mathtt{y} : B \vdash M : C \quad \Gamma, \mathtt{x} : D \vdash M' : E}{\Gamma \vdash \lambda\{(\#\mathsf{l}, \mathtt{x}, \mathtt{y}).M, (\#\mathsf{r}, \mathtt{x}).M'\} : \boxed{\textstyle\prod}\{\#\mathsf{l}.A,\ B \vdash C; \#\mathsf{r}.D \vdash E\}}$$

and two elimination rules for application:

$$\frac{\Gamma \vdash M : \boxed{\textstyle\prod}\{\#\mathsf{l}.A,\ B \vdash C; \#\mathsf{r}.D \vdash E\} \quad \Gamma \vdash N : A \quad \Gamma \vdash N' : B}{\Gamma \vdash M(\#\mathsf{l}, N, N') : C}$$

$$\frac{\Gamma \vdash M : \boxed{\textstyle\prod}\{\#\mathsf{l}.A,\ B \vdash C; \#\mathsf{r}.D \vdash E\} \quad \Gamma \vdash N : D}{\Gamma \vdash M(\#\mathsf{r}, N) : E}$$

It is straightforward to give CBV and CBN operational semantics for all these terms. For recursion, we follow Fig. 7; in the case of CBV, we allow recursive λ-abstractions of type $\boxed{\prod}\{\#\mathsf{l}.A,\ B \vdash C; \#\mathsf{r}.D \vdash E\}$.

In the absence of effects, $\boxed{\sum}$ and $\boxed{\prod}$ could be seen as mere syntactic sugar, built up from the connectives 0, $+$, 1, $\times$, $\to$. But it is argued in [22] that this viewpoint falls down in the effectful setting, because many isomorphisms cease to hold.

It is easy to see how to generalize these two prototypical examples to arbitrary tuple types and arbitrary function types; the details are given in [22], and the resulting form of λ-calculus is called *jumbo* λ-calculus. We accordingly assume the CBV and CBN source languages to be jumbo λ-calculus extended with effects. All the connectives we have seen so far, viz. $\mathtt{bool}$, $+$, $\to$, as well as the n-ary function types mentioned in Section 2.2, are instances of these connectives.

The translation of our two example connectives into CBPV is shown in Fig. 10.

We translate the general connectives into CBPV in the same way we translated the two examples, and adapt the proofs of Proposition 7–11 accordingly.

Proposition 12 (junk-freeness and full abstraction).

CBN

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

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

CBV

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

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

$A_0, \ldots, A_{n-1} \vdash V : C \quad \text{(value)}$	$A_0^{\mathsf{v}}, \ldots, A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{v}} V^{\mathsf{val}} : C^{\mathsf{v}}$
$\langle \#\mathsf{l}, V, V' \rangle$	$\langle \#\mathsf{l}, \langle V^{\mathsf{val}}, V'^{\mathsf{val}}\rangle\rangle$
$\langle \#\mathsf{r}, V \rangle$	$\langle \#\mathsf{r}, V^{\mathsf{val}}\rangle$
$\lambda\{(\#\mathsf{l}, \mathsf{x}, \mathsf{y}).M, (\#\mathsf{r}, \mathsf{x}).M'\}$	$\mathtt{thunk}\ \lambda\{\#\mathsf{l}.\lambda\mathsf{x}.\lambda\mathsf{y}.M^{\mathsf{v}}, \#\mathsf{r}.\lambda\mathsf{x}.M'^{\mathsf{v}}\}$
$\mathtt{rec}\ \mathsf{f}\ \lambda\{(\#\mathsf{l}, \mathsf{x}, \mathsf{y}).M, (\#\mathsf{r}, \mathsf{x}).M'\}$	$\mathtt{thunk}\ \mathtt{rec}\ \mathsf{f}\ \lambda\{\#\mathsf{l}.\lambda\mathsf{x}.\lambda\mathsf{y}.M^{\mathsf{v}}, \#\mathsf{r}.\lambda\mathsf{x}.M'^{\mathsf{v}}\}$

Fig. 10 Translating example connectives from CBN and CBV into CBPV

1. *The translation from CBN jumbo λ-calculus with any of our effects (divergence, printing or storage) to CBPV with the same effects is*

   junk-free *i.e. for each computation* $U A_0^{\mathsf{n}}, \ldots, U A_{n-1}^{\mathsf{n}} \vdash M : B^{\mathsf{n}}$ *in the target language, there is a term* $A_0, \ldots, A_{n-1} \vdash N : B$ *in the source language such that* $N^{\mathsf{n}} = M$ *is provable in the CBPV theory*
   fully abstract *i.e.* $M \lesssim_{\mathsf{CBN}} N$ *iff* $M^{\mathsf{n}} \lesssim N^{\mathsf{n}}$.

2. *The translation from CBV jumbo λ-calculus with any of our effects (divergence, printing or storage) to CBPV with the same effects is*

   junk-free *i.e. for each computation* $A_0^{\mathsf{v}}, \ldots, A_{n-1}^{\mathsf{v}} \vdash M : FB^{\mathsf{v}}$ *in the target language, there is a term* $A_0, \ldots, A_{n-1} \vdash N : B$ *in the source language such that* $N^{\mathsf{v}} = M$ *is provable in the CBPV theory*
   fully abstract *i.e.* $M \lesssim_{\mathsf{CBV}} N$ *iff* $M^{\mathsf{v}} \lesssim N^{\mathsf{v}}$.

We omit the proof of junk-freeness, which uses a reverse translation from CBPV to CBN and from CBPV to CBV, and is given in detail in [20]. Full abstraction follows from junk-freeness in the standard manner [30].

The question remains whether these results hold for *smaller* source languages. In some cases, affirmative results can be obtained by showing that the smaller source languages provide enough connectives to *define* the general connectives of jumbo λ-calculus. In other cases—such as the one mentioned above, where the source language is CBN without ternary sum—the answer is unknown. But it seems likely affirmative answers for various effects can be obtained from junk-freeness results for suitable denotational semantics, such as game semantics.

7.6 Untyped languages

The simulation results Proposition 6–9 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, the traditional observation in untyped CBN λ-calculus, if one wishes the η-law to be valid, is *reduction to head normal form* [2]. That is unsuited to typed languages, e.g. it would distinguish the CBN terms $\lambda \mathtt{x}.\mathtt{x}$ and $\lambda \mathtt{x}.\mathtt{diverge}$ of type $1 \to 1$, even though they have the same denotation.

8 Complex values and the CBPV equational theory

8.1 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 [7].

The CBPV equational theory is the minimal congruence containing the laws in Fig. 11 (with R ranging over computations). 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

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

$$\begin{array}{rcl}
\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ R & = & R[V/\mathtt{x}] \\
(\mathtt{return}\ V)\ \mathtt{to}\ \mathtt{x}.\ M & = & M[V/\mathtt{x}] \\
\mathtt{force}\ \mathtt{thunk}\ M & = & M \\
\mathtt{pm}\ \langle i, V\rangle\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.R_i\}_{i\in I} & = & R_{\hat{i}}[V/\mathtt{x}] \\
\mathtt{pm}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.R & = & R[V/\mathtt{x}, V'/\mathtt{y}] \\
\hat{i}\text{`}\lambda\{i.M_i\}_{i\in I} & = & M_{\hat{i}} \\
V\text{`}\lambda\mathtt{x}.M & = & M[V/\mathtt{x}]
\end{array}$$

η-laws

$$\begin{array}{rcl}
M & = & M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{return}\ \mathtt{x} \\
V & = & \mathtt{thunk}\ \mathtt{force}\ V \\
R[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ \langle i, \mathtt{x}\rangle.\{{}^{\mathtt{x}}R[\langle i, \mathtt{x}\rangle/\mathtt{z}]\}_{i\in I} \\
R[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ {}^{\mathtt{xy}}R[\langle \mathtt{x}, \mathtt{y}\rangle/\mathtt{z}] \\
M & = & \lambda\{i.i\text{`}M\}_{i\in I} \\
M & = & \lambda\mathtt{x}.(\mathtt{x}\text{`}\ {}^{\mathtt{x}}M)
\end{array}$$

sequencing laws

$$\begin{array}{rcl}
(P\ \mathtt{to}\ \mathtt{x}.\ M)\ \mathtt{to}\ \mathtt{y}.\ N & = & P\ \mathtt{to}\ \mathtt{x}.\ (M\ \mathtt{to}\ \mathtt{y}.\ {}^{\mathtt{x}}N) \\
P\ \mathtt{to}\ \mathtt{x}.\ \lambda\{i.M_i\}_{i\in I} & = & \lambda\{i.(P\ \mathtt{to}\ \mathtt{x}.\ M_i)\}_{i\in I} \\
P\ \mathtt{to}\ \mathtt{x}.\ \lambda\mathtt{y}.M & = & \lambda\mathtt{y}.({}^{\mathtt{y}}P\ \mathtt{to}\ \mathtt{x}.\ M)
\end{array}$$

print laws

$$\begin{array}{rcl}
(\mathtt{print}\ c.\ M)\ \mathtt{to}\ \mathtt{x}.\ N & = & \mathtt{print}\ c.\ (M\ \mathtt{to}\ \mathtt{x}.\ N) \\
\mathtt{print}\ c.\ \lambda\{i.M_i\}_{i\in I} & = & \lambda\{i.\mathtt{print}\ c.\ M\}_{i\in I} \\
\mathtt{print}\ c.\ \lambda\mathtt{x}.M & = & \lambda\mathtt{x}.\mathtt{print}\ c.\ M
\end{array}$$

diverge laws

$$\begin{array}{rcl}
\mathtt{diverge}\ \mathtt{to}\ \mathtt{x}.\ N & = & \mathtt{diverge} \\
\mathtt{diverge} & = & \lambda\{i.\mathtt{diverge}\}_{i\in I} \\
\mathtt{diverge} & = & \lambda\mathtt{x}.\mathtt{diverge}
\end{array}$$

Fig. 11 CBPV equations

$\mathtt{x}$ bound to V". Similarly the η-law for functions says: "if we pop $\mathtt{x}$, then push $\mathtt{x}$, then do M which ignores $\mathtt{x}$, that is the same as doing M".

Proposition 13 (soundness of equational theory). *The equations in Fig.* 11 *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.*

8.2 Complex values

The CBPV typing rules presented in Fig. 2 allow computations to be formed by pattern-matching, but not values. As stated for fine-grain CBV in Section 2.2, this keeps the operational semantics simple. On the other hand, certain desirable values and equations between values are missing. For example, we can see that

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

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

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

Fig. 12 Complex values

1. there is no value $\mathtt{x} : 0 \times 0 \vdash^{\mathsf{v}} V : 0$
2. the equation $\mathtt{x} : 0 \vdash^{\mathsf{v}} \mathtt{true} = \mathtt{false} : 1 + 1$ (where `true` and `false` abbreviate $\langle \#\mathsf{l}, \langle\rangle\rangle$ and $\langle \#\mathsf{r}, \langle\rangle\rangle$ respectively) cannot be proved using the laws of Fig. 11.

In the remainder of the paper, we wish to prove the correspondence of CBPV with a categorical semantics and with Effect-PCF, and, for these purposes, we have to rectify these problems. To do so, we add *complex values*, whose syntax is displayed in Fig. 12. The equational theory on CBPV with complex values is the least congruence containing the laws of Fig. 11, with R ranging over all terms (values and computations).

More generally, we define *a* theory on CBPV with complex values to be a substitutive congruence containing these laws. (The least congruence is automatically substitutive.) In the sequel, τ ranges over theories with complex values, and σ over theories without complex values. In the latter case, we write $\sigma_{+\mathsf{CV}}$ for the least extension of σ to a theory with complex values. We also write $\vdash^{\mathsf{c}}_{+\mathsf{CV}}$ and $\vdash^{\mathsf{v}}_{+\mathsf{CV}}$ to indicate the syntax and least theory of CBPV with complex values, and $\vdash^{\mathsf{c}}_{-\mathsf{CV}}$ and $\vdash^{\mathsf{v}}_{-\mathsf{CV}}$ to indicate the syntax and least theory of CBPV without complex values.

Complex values can be regarded as a minor addition to the language, in the sense that they have no impact on computations:

Proposition 14 (complex values do not affect computations).

1. *(definability) For any computation* $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} M : \underline{B}$*, there is a computation* $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} N : \underline{B}$ *such that* $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} M = N : \underline{B}$.
2. *(conservativity) Let* σ *be an equational theory on CBPV without complex values (i.e. a congruence containing all the laws of Fig. 11). Let* $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} N, N' : \underline{B}$ *be* computations. *Then* $N = N'$ *is in* $\sigma_{+\mathsf{CV}}$ *iff* $N = N'$ *in* σ.

Examples (1)–(2) *above show that these results do not hold for values.*

Proof: Following [8], we define a *thunkable* from Γ to B to be a computation $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} M : FB$ such that

$$\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \mathtt{return\ thunk}\ M = M\ \mathtt{to\ x.\ return\ thunk\ return\ x} : FUFB \tag{9}$$

Condition (9) is equivalent to the following: for any context $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \mathcal{C}[\cdot] : \underline{B}$ that does not bind any identifiers, we have

$$\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \mathcal{C}[M] = M\ \mathtt{to\ x.}\ \mathcal{C}[\mathtt{return\ x}] : \underline{B} \tag{10}$$

$\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} M : \underline{B}$	$\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{M} : \underline{B}$
$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{let}\ \mathtt{z}\ \mathtt{be}\ \mathtt{x}.\ \tilde{M}$
$\mathtt{return}\ V$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{return}\ \mathtt{z}$
$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\tilde{M}\ \mathtt{to}\ \mathtt{x}.\ \tilde{N}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.M_i\}_{i\in I}$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\tilde{M}_i\}_{i\in I}$
$\mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.M$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\tilde{M}$
$\lambda\{i.M_i\}_{i\in I}$	$\lambda\{i.\tilde{M}_i\}_{i\in I}$
$\hat{\imath}\text{`}M$	$\hat{\imath}\text{`}\tilde{M}$
$\lambda\mathtt{x}.M$	$\lambda\mathtt{x}.\tilde{M}$
$V\text{`}M$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{z}\text{`}\tilde{M}$
$\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : B$	$\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{V} : FB$
$\mathtt{x}$	$\mathtt{return}\ \mathtt{x}$
$\langle \hat{\imath}, V\rangle$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{return}\ \langle \hat{\imath}, \mathtt{z}\rangle$
$\langle V, V'\rangle$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \tilde{V}'\ \mathtt{to}\ \mathtt{z}'.\ \mathtt{return}\ \langle \mathtt{z}, \mathtt{z}'\rangle$
$\mathtt{thunk}\ M$	$\mathtt{return}\ \mathtt{thunk}\ \tilde{M}$
$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ W$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{let}\ \mathtt{z}\ \mathtt{be}\ \mathtt{x}.\ \tilde{W}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.W_i\}_{i\in I}$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\tilde{W}_i\}_{i\in I}$
$\mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.W$	$\tilde{V}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\tilde{W}$

Fig. 13 Definitions used in the proof of Proposition 14

The equivalence follows from the fact that

$$\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \mathcal{C}[M] = (\mathtt{return}\ \mathtt{thunk}\ M)\ \mathtt{to}\ \mathtt{y}.\ \mathcal{C}[\mathtt{force}\ \mathtt{y}] : \underline{B}$$

It is easy to show that every *safe* computation, in the sense of Definition 3, is thunkable.

In Fig. 13, we define compositionally

- for each computation $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} M : \underline{B}$, a computation $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{M} : \underline{B}$ such that $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} \tilde{M} = M : \underline{B}$ is provable.
- for each value $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : B$ in CBPV, a safe (and hence thunkable) computation $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{V} : FB$ such that $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} \tilde{V} = \mathtt{return}\ V : FB$ is provable.

This proves definability. For conservativity, we first show that if $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} W : A$ then

$$\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} M[\widetilde{W}/\mathtt{x}] = \tilde{W}\ \mathtt{to}\ \mathtt{x}.\ \tilde{M} : \underline{B}$$

$$\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} V[\widetilde{W}/\mathtt{x}] = \tilde{W}\ \mathtt{to}\ \mathtt{x}.\ \tilde{V} : FB$$

by mutual induction on $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}}_{+\mathsf{CV}} M : \underline{B}$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : B$. We deduce, by induction, that

- if $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} M = M' : \underline{B}$ then $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{M} = \tilde{M}' : \underline{B}$
- if $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} V = V' : B$ then $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{V} = \tilde{V}' : FB$

We also show that

- if $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} N : \underline{B}$ then $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{N} = N : \underline{B}$ is provable
- if $\Gamma \vdash^{\mathsf{v}}_{-\mathsf{CV}} W : B$ then $\Gamma \vdash^{\mathsf{c}}_{-\mathsf{CV}} \tilde{W} = \mathtt{return}\ W : FA$ is provable.

Given σ, we prove

$\Gamma, \Delta \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : B$		$\Gamma \vdash^{\mathsf{v}}_{-\mathsf{CV}} \acute{V}(\overrightarrow{W/\mathtt{y}}) : B$
$\mathtt{x}$	declared in Γ	$\mathtt{x}$
$\mathtt{y}_i$	declared in Δ	W_i
$\langle \hat{\imath}, V \rangle$		$\langle \hat{\imath}, \acute{V}(\overrightarrow{W/\mathtt{y}}) \rangle$
$\langle V, V' \rangle$		$\langle \acute{V}(\overrightarrow{W/\mathtt{y}}), \acute{V}'(\overrightarrow{W/\mathtt{y}}) \rangle$
$\mathtt{thunk}\ M$		$\overbrace{M[\overrightarrow{W/\mathtt{y}}]}$
$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ V'$	where $\acute{V}(\overrightarrow{W/\mathtt{y}}) = W$	$\acute{V}'(\overrightarrow{W/\mathtt{y}}, W/\mathtt{x})$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle . V_i\}_{i \in I}$	where $\acute{V}(\overrightarrow{W/\mathtt{y}}) = \langle \hat{\imath}, W \rangle$	$\acute{V}_{\hat{\imath}}(\overrightarrow{W/\mathtt{y}}, W/\mathtt{x})$
$\mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{x}' \rangle . V'$	where $\acute{V}(\overrightarrow{W/\mathtt{y}}) = \langle W, W' \rangle$	$\acute{V}'(\overrightarrow{W/\mathtt{y}}, W/\mathtt{x}, W'/\mathtt{x}')$

Fig. 14 Definitions used in the proof of Proposition 15

– if $M = M'$ in $\sigma_{+\mathsf{CV}}$ then $\tilde{M} = \tilde{M}'$ in σ
– if $V = V'$ in $\sigma_{+\mathsf{CV}}$ then $\tilde{V} = \tilde{V}'$ in σ

by induction, and the result follows. □

Proposition 14(1) enables us to "evaluate" a computation with complex 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.

Although Proposition 14 does not apply to values, we have some limited results as follows.

Proposition 15. *Let* Γ *be a context that is* tuple-free*, i.e. no identifier in it has* $\sum, 1, \times$ *type. For any value* $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : B$*, there exists* $\Gamma \vdash^{\mathsf{v}}_{-\mathsf{CV}} W : B$ *such that* $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} V = W : B$ *is provable.*

Proof: Fix Γ. Given a value $\Gamma, \Delta \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : B$, and, for each identifier $(\mathtt{y}_i : B) \in \Delta$, a value $\Gamma \vdash^{\mathsf{v}}_{-\mathsf{CV}} W_i : B$, we define, in Fig. 14, a "pseudo-substitution" $\Gamma \vdash^{\mathsf{v}}_{-\mathsf{CV}} \acute{V}(\overrightarrow{W/\mathtt{y}}) : B$ such that $\Gamma \vdash^{\mathsf{v}}_{-\mathsf{CV}} \acute{V}(\overrightarrow{W/\mathtt{y}}) = V[\overrightarrow{W/\mathtt{y}}] : B$ is provable. (The penultimate clause exploits the fact that any value $\Gamma \vdash^{\mathsf{v}}_{-\mathsf{CV}} V : \sum_{i \in I} A_i$ must be of the form $\langle \hat{\imath}, W \rangle$, rather than an identifier, because Γ is tuple-free. Similarly for the last clause.) Finally, given $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : B$, we define W to be $\acute{V}$ applied to the empty pseudo-substitution. □

Definition 4. A theory σ (with or without complex values) is *inconsistent* when $\vdash^{\mathsf{c}} \mathtt{return}\ \mathtt{true} = \mathtt{return}\ \mathtt{false} : F(1+1)$ in σ, or, equivalently, when $\Gamma \vdash^{\mathsf{c}} M = N : \underline{B}$ is in σ for all $\Gamma \vdash^{\mathsf{c}} M, N : \underline{B}$. (Thus values are not necessarily equated.) Otherwise it is *consistent*.

Proposition 16. *Closed values within a consistent theory have the following properties.*

1. *Let* σ *be a consistent theory without complex values. For closed values* $\vdash^{\mathsf{c}}_{-\mathsf{CV}} W, W' : B$*, if* $\mathtt{return}\ W = \mathtt{return}\ W'$ *is in* σ*, then* $W = W'$ *is in* σ*.*

2. *Let σ be a consistent theory without complex values. For closed values $\vdash^{\mathsf{v}}_{-\mathsf{CV}} W, W' : B$, if $W = W'$ is in $\sigma_{+\mathsf{CV}}$, then it is in σ.*
3. *Let τ be a consistent theory with complex values. For closed values $\vdash^{\mathsf{c}}_{+\mathsf{CV}} V, V' : B$, if $\mathtt{return}\ V = \mathtt{return}\ V'$ is in τ, then $V = V'$ is in τ.*

Proof: We prove these in the stated order.

1. This is by induction on W. If W has type $U\underline{B}$, it follows from

$$W = \mathtt{thunk}\,(\mathtt{return}\ W\ \mathtt{to}\ \mathtt{x}.\ \mathtt{force}\ \mathtt{x})$$

   If W is $\langle \hat{\imath}, V\rangle$ and W' is $\langle \hat{\imath}', V'\rangle$ then, firstly we apply the context

$$[\cdot]\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \left\{\begin{array}{l}\langle \hat{\imath}, \mathtt{x}\rangle.\ \mathtt{return}\ \mathtt{true}\\ \langle i, \mathtt{x}\rangle.\ \mathtt{return}\ \mathtt{false}\ (i \neq \hat{\imath})\end{array}\right\}$$

   to $\mathtt{return}\ W = \mathtt{return}\ W'$, which if $\hat{\imath} \neq \hat{\imath}'$ gives $\mathtt{return}\ \mathtt{true} = \mathtt{return}\ \mathtt{false}$ in σ. Since σ is consistent, we have $\hat{\imath} = \hat{\imath}'$. Then we apply the context

$$[\cdot]\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \left\{\begin{array}{l}\langle \hat{\imath}, \mathtt{x}\rangle.\ \mathtt{return}\ \mathtt{x}\\ \langle i, \mathtt{x}\rangle.\ \mathtt{return}\ v\ (i \neq \hat{\imath})\end{array}\right\}$$

   to $\mathtt{return}\ W = \mathtt{return}\ W'$, giving $\mathtt{return}\ V = \mathtt{return}\ V'$ in σ, so $V = V'$ is in σ by the inductive hypothesis. Hence $W = W'$ is in σ. The case where W is $\langle V, V'\rangle$ is similar.
2. We have $\mathtt{return}\ W = \mathtt{return}\ W'$ in $\sigma_{+\mathsf{CV}}$, and hence also, by Proposition 14(14), in σ. By Proposition 16(1), we deduce that $W = W'$ is in σ.
3. Using Proposition 15, we obtain $\vdash^{\mathsf{v}}_{-\mathsf{CV}} W, W' : B$ such that $\vdash^{\mathsf{v}}_{+\mathsf{CV}} V = W : B$ and $\vdash^{\mathsf{v}}_{+\mathsf{CV}} V' = W' : B$ are provable. Then $\mathtt{return}\ W = \mathtt{return}\ W'$ is in τ. Since the restriction of τ to complex-value-free terms is consistent, Proposition 16(1) tells us that $W = W'$ is in τ, and so $V = V'$ is in τ.

□

9 Every model is equivalent to an algebra model

In Section 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 5. 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 $X \in \mathtt{ob}\,\mathcal{C}$, an index $FX \in \mathfrak{J}$ mapped by K to the free algebra on X

exponential algebras for each $X \in \mathtt{ob}\,\mathcal{C}$ and index $\underline{Y} \in \mathfrak{J}$, an exponential E from X 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 Definition 5 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 Definition 5.

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[6].

Definition 6 (object structures).

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.
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 [12,16], which is applicable to many simply typed calculi.

Definition 7. Let τ be a CBPV object structure.

1. A *τ-sequent* Q is either

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

   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* s 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.

[6] 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.

Definition 8. 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 Figs. 2 and 12 together with the rules

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

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

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. 11. 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 a 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 9. 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 17. *Let τ be a CBPV object structure. Then 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

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

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

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

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

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

10 Comparison with Filinski's monadic metalanguage and Marz's SFPL

Having treated complex values in some detail, we are in a position to look closely at the relationship between

– CBPV

Effect-PCF

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

$$\text{judgement} \qquad A_0, \ldots, A_{n-1} \vdash B$$

SFPL

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

$$\text{judgement} \qquad A_0, \ldots, A_{n-1} \vdash \underline{B}$$

Fig. 15 Types and judgements of Effect-PCF and SFPL

– Effect-PCF, a version of the monadic metalanguage appearing in [7]
– SFPL [24], a variant of the earlier language SFL [23].

We treat Effect-PCF in detail, because the relationship between CBPV and monads is a central theme of this paper; but we treat SFPL in outline only. We omit recursive types, as these are beyond the scope of this paper. The non-recursive types and the judgements of the two languages are given[7] in Fig. 15. The syntax of Effect-PCF is given in Fig. 16. We have modified syntax slightly to agree with CBPV, in particular using M `to` `x`. N for sequencing in Effect-PCF.

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 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 $A_0, \ldots, A_{n-1} \vdash^{\mathrm{c}} M : \underline{B}$ denotes a strict function from $[\![A_0]\!] \otimes \cdots \otimes [\![A_{n-1}]\!]$ to $[\![\underline{B}]\!]$. This semantics uses smash 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:

$$\overline{U\underline{B}} = \overline{\underline{B}}$$
$$\overline{FA} = T\overline{A}$$

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

[7] Caution: [24] uses the phrase "computational types" for what we have called the "value types" of SFPL.

Primitives

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

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

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

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

$$\frac{\Gamma \vdash M_i : \underline{B}_i \ \ (\forall i \in I)}{\Gamma \vdash \lambda \{i.M_i\}_{i \in I} : \prod_{i \in I} \underline{B}_i} \qquad \frac{\Gamma \vdash N : \prod_{i \in I} \underline{B}_i}{\Gamma \vdash \hat{\imath}\text{`}N : \underline{B}_{\hat{\imath}}}$$

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

Derived

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

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

$$\begin{aligned} M\ \mathtt{to\ x}.^{TA}\ N &= M\ \mathtt{to\ x}.\ N \\ M\ \mathtt{to\ x}.^{\prod_{i \in I} \underline{B}_i}\ N &= \lambda \{i.(M\ \mathtt{to\ x}.^{\underline{B}_i}\ (i\text{`}N))\}_{i \in I} \\ M\ \mathtt{to\ x}.^{A \to \underline{B}}\ N &= \lambda \mathtt{y}.(M\ \mathtt{to\ x}.^{\underline{B}}\ (\mathtt{y}\text{`}N)) \end{aligned}$$

Fig. 16 Terms of Effect-PCF (slightly modified)

defined by mutual induction:

$$\begin{aligned} \widetilde{\underline{B}} &= U\widehat{\underline{B}} \\ \widehat{TA} &= F\widetilde{A} \end{aligned}$$

and all the other clauses are trivial. It is obvious that $\overline{\ }$ on value types is inverse to $\widetilde{\ }$, and $\overline{\ }$ on computation types is inverse to $\widehat{\ }$. In the same way, we can define a bijection between CBPV value (resp. computation) types and SFPL value (resp. computation) types.

Proceeding to terms, the sole judgement of Effect-PCF corresponds to $\vdash^{\mathrm{v}}$ in CBPV. Hence we translate every Effect-PCF term into a CBPV value, as shown in Fig. 17. This translation preserves provable equality. Moreover, the equation

$$\widetilde{M\ \mathtt{to\ x}.^{\underline{B}}\ N} = \mathtt{thunk}\ ((\mathtt{force}\ \widetilde{M})\ \mathtt{to\ x}.\ \mathtt{force}\ \widetilde{N})$$

is provable in the equational theory with complex values; this is shown by induction on the type $\underline{B}$ of N.

In the opposite direction, we define a translation $\overline{\ }$ taking

$A_0, \ldots, A_{n-1} \vdash M : B$	$\widetilde{A_0}, \ldots, \widetilde{A_{n-1}} \vdash^{\mathsf{v}}_{+\mathsf{CV}} \widetilde{M} : \widetilde{B}$
$\mathtt{x}$	$\mathtt{x}$
$\mathtt{let}\ M\ \mathtt{be}\ \mathtt{x}.\ N$	$\mathtt{let}\ \widetilde{M}\ \mathtt{be}\ \mathtt{x}.\ \widetilde{N}$
$\langle \hat{\imath}, M\rangle$	$\langle \hat{\imath}, \widetilde{M}\rangle$
$\mathtt{pm}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.N_i\}_{i\in I}$	$\mathtt{pm}\ \widetilde{M}\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\widetilde{N_i}\}_{i\in I}$
$\langle M, M'\rangle$	$\widetilde{\langle M, M'\rangle}$
$\mathtt{pm}\ M\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ N$	$\mathtt{pm}\ \widetilde{M}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ \widetilde{N}$
$\mathtt{return}\ M$	$\mathtt{thunk}\ \mathtt{return}\ \widetilde{M}$
$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\mathtt{thunk}\ ((\mathtt{force}\ \widetilde{M})\ \mathtt{to}\ \mathtt{x}.\ \mathtt{force}\ \widetilde{N})$
$\lambda \mathtt{x}.M$	$\mathtt{thunk}\ \lambda \mathtt{x}.\ \mathtt{force}\ \widetilde{M}$
$N \text{`} M$	$\mathtt{thunk}\ (\widetilde{N} \text{`} \mathtt{force}\ \widetilde{M})$
$\lambda\{i.M_i\}_{i\in I}$	$\mathtt{thunk}\ \lambda\{i.\ \mathtt{force}\ \widetilde{M_i}\}_{i\in I}$
$\hat{\imath} \text{`} M$	$\mathtt{thunk}\ (\hat{\imath} \text{`} \mathtt{force}\ \widetilde{M})$

Fig. 17 Translation from Effect-PCF terms to CBPV values

– 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:

$$\overline{\mathtt{thunk}\ M} = \overline{M}$$
$$\overline{\mathtt{force}\ V} = \overline{V}$$
$$\overline{M\ \mathtt{to}\ \mathtt{x}.\ N} = \overline{M}\ \mathtt{to}\ \mathtt{x}.^{\overline{\underline{B}}}\ \overline{N} \quad \text{where } \underline{B} \text{ is the type of } N$$

and all the other clauses are trivial. This preserves provable equality, because the analogues of all the CBPV laws are provable in Effect-PCF (using induction over types for the sequencing laws).

Finally we prove

– for any Effect-PCF term $\Gamma \vdash M : B$, the equation $\Gamma \vdash \widetilde{\overline{M}} = M : B$ is provable
– for any CBPV value $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} V : A$, the equation $\Gamma \vdash^{\mathsf{v}}_{+\mathsf{CV}} \widetilde{\overline{V}} = V : B$ is provable
– for any CBPV computation $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} M : \underline{B}$, the equation $\Gamma \vdash^{\mathsf{c}}_{+\mathsf{CV}} \mathtt{force}\ \widetilde{\overline{M}} = M : \underline{B}$ is provable.

by induction on the terms. Hence the translations reflect provable equality.

As for SFPL, its sole judgement corresponds to $\vdash^{\mathsf{c}}$ in CBPV. The relationship between CBPV and SFPL terms is somewhat similar to the above, although not as tight. We omit details.

11 Conclusions

We summarize the advances represented by call-by-push-value.

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 Section 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 [27], where $\underline{A} \to_{\mathbf{CBN}} \underline{B}$ denotes $(S \to [\![\underline{A}]\!]) \to [\![\underline{B}]\!]$ 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 [33], 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 a computational way.

As stated in Section 1.1, this paper is an introduction to CBPV, not an exhaustive study. In particular, the relationship between CBPV and adjunctions [19,21] is not investigated in this paper. However, in the particular models we have studied, it is quite apparent that U and F represent an adjunction.

- 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 [20]. Furthermore, it remains to compare this work to the line of research in [15,32]; this is closely related to continuation semantics, which we have not included in this paper.

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 [34].

Here is the proof of Proposition 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

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

The definition of these subsets proceeds by induction over types:

$$
\begin{array}{ll}
\mathtt{thunk}\, M \in \mathsf{red}_{U\underline{B}} & \text{iff } s, M \in \mathsf{red}_{\underline{B}} \text{ for all } s \in S \\
\langle \hat{\imath}, V \rangle \in \mathsf{red}_{\sum_{i\in I} A_i} & \text{iff } V \in \mathsf{red}_{A_{\hat{\imath}}} \\
\langle V, V' \rangle \in \mathsf{red}_{A\times A'} & \text{iff } V \in \mathsf{red}_A \text{ and } V' \in \mathsf{red}_{A'} \\
s, M \in \mathsf{red}_{FA} & \text{iff } s, M \Downarrow s', \mathtt{return}\, V \text{ where } V \in \mathsf{red}_A \\
s, M \in \mathsf{red}_{\prod_{i\in I} \underline{B}_i} & \text{iff } s, M \Downarrow s', \lambda\{i.M_i\}_{i\in I} \text{ where } i \in I \text{ implies } s', M_i \in \mathsf{red}_{\underline{B}_i} \\
s, M \in \mathsf{red}_{A\to \underline{B}} & \text{iff } s, M \Downarrow s', \lambda \mathtt{x}.N \text{ where } V \in \mathsf{red}_A \text{ implies } s', M[V/\mathtt{x}] \in \mathsf{red}_{\underline{B}}
\end{array}
$$

We note that $s, M \in \mathsf{red}_{\underline{B}}$ iff $s, M \Downarrow s', T$ for some $s', T \in \mathsf{red}_{\underline{B}}$.

Finally we show that for any computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, if $s \in S$ and $W_i \in \mathsf{red}_{A_i}$ for $i = 0, \ldots, n-1$ then $s, M[\overrightarrow{W_i/\mathtt{x}_i}] \in \mathsf{red}_{\underline{B}}$; 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 Proposition 2.

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

- for each value type A, a relation $\leqslant_A$ between $[\![A]\!]$ and closed values of type A such that, for each V, the set $\{a \mid a \leqslant_A V\}$ is admissible and down-closed
- for each computation type $\underline{B}$ a relation $\leqslant_{\underline{B}}$ between $[\![\underline{B}]\!]$ and closed computations of type $\underline{B}$, such that, for each M, the set $\{a \mid a \leqslant_{\underline{B}} M\}$ is admissible, down-closed and $\bot$-containing.

(Our proof does not make use of the down-closure property.) The definition of these relations proceeds by induction over types:

$$
\begin{array}{lll}
a \leqslant_{U\underline{B}} \mathtt{thunk}\, M & \text{iff} & a \leqslant_{\underline{B}} M \\
a \leqslant_{\sum_{i\in I} A_i} \langle \hat{\imath}, V\rangle & \text{iff} & a = \langle \hat{\imath}, b\rangle \text{ for some } b \leqslant_{A_{\hat{\imath}}} V \\
a \leqslant_{A\times A'} \langle V, V'\rangle & \text{iff} & a = \langle b, b'\rangle \text{ for some } b \leqslant_A V \text{ and } b' \leqslant_{A'} V' \\
 & & \\
b \leqslant_{FA} M & \text{iff} & b = \bot \text{ or} \\
 & & \quad b = \mathsf{up}\, a \text{ and } M \Downarrow \mathtt{return}\, V \text{ and } a \leqslant_A V \\
f \leqslant_{\prod_{i\in I}\underline{B}_i} M & \text{iff} & f = \bot \text{ or} \\
 & & \quad M \Downarrow \lambda\{i.N_i\}_{i\in I}, \text{ and } i \in I \text{ implies } f_{\hat{\imath}} \leqslant_{\underline{B}_i} N_i \\
f \leqslant_{A\to\underline{B}} M & \text{iff} & f = \bot \text{ or} \\
 & & \quad M \Downarrow \lambda\mathtt{x}.N, \text{ and } a \leqslant_A V \text{ implies } f a \leqslant_{\underline{B}} N[V/\mathtt{x}]
\end{array}
$$

We note that $b \leqslant_{\underline{B}} M$ iff either $b = \bot$ or $M \Downarrow T$ for some terminal T such that $b \leqslant_{\underline{B}} T$.

Finally, we show that for any computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, if $a_i \leqslant_{A_i} W_i$ for $i = 0, \ldots, n-1$ then $[\![M]\!]\overrightarrow{\mathtt{x}_i \mapsto a_i} \leqslant_{\underline{B}} M[\overrightarrow{W_i/\mathtt{x}_i}]$; 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. □

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

References

1. Abramsky, S.: The lazy λ-Calculus. In Research Topics in Functional Programming. Addison Wesley, pp. 65–117 (1990)
2. Barendregt, H.: The Lambda-Calculus: Its Syntax and Semantics. North-Holland, Amsterdam (1980)
3. Benton, N., Hughes, J., Moggi, E.: Monads and effects. In: Barthe, G., Dybjer, P., Pinto, L., Saraiva, J. (eds.), Advanced Lectures From International Summer School on Applied Semantics (APPSEM), Caminha, Portugal, vol. 2395 of Lecture Notes in Computer Science, pp. 42–122 (2000)

4. Benton, N., Kennedy, A.: Monads, effects and transformations. In: Gordon, A., Pitts, A. (eds.), Proceedings, Higher-Order Operational Techniques in Semantics (HOOTS '99), vol. 26 of *ENTCS*, Paris, France, pp. 3–20 (1999)
5. Benton, N., Wadler, P.: Linear logic, monads and the lambda calculus. In: Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, IEEE Computer Society Press, pp. 420–431 (1996)
6. Felleisen, M., Friedman, D.: Control operators, the SECD-machine, and the λ-calculus. In: Wirsing, M. (ed.), Formal Description of Programming Concepts III, North-Holland, pp. 193–217 (1986)
7. Filinski, A.: Controlling Effects. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania (1996)
8. Führmann, C.: Direct models for the computational λ-calculus. In: Brookes, S., Jung, A., Mislove, M., Scedrov, A. (eds.), In: Proceedings of the 15th Conference in Mathematical Foundations of Programming Semantics, New Orleans, vol. 20 of *ENTCS*, pp. 147–172 (1999).
9. Hatcliff, J.: The Structure of Continuation-Passing Styles. PhD thesis, Kansas State University, (1994)
10. Hatcliff, J., Danvy, O.: Thunks and the λ-calculus. Journal of Functional Programming, **7**(3), 303–319 (1997)
11. Howard, B.: Inductive, coinductive, and pointed types. In: Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '96), vol. 31, no. 6 of *ACM SIGPLAN Notices*, pp. 102–109. ACM, (1996)
12. Jeffrey, A.: A fully abstract semantics for a higher-order functional language with nondeterministic computation. Theoretical Computer Science, **228**(1–2), 105–150 (1999)
13. Krivine, J.-L.: Un interpréteur de λ-calcul. Unpublished, (1985)
14. Laird, J.: A Semantic Analysis of Control. PhD thesis, University of Edinburgh (1998)
15. Laurent, O.: Polarized proof-nets: proof-nets for LC (extended abstract). In: Girard, J.-Y. (ed.), Typed Lambda Calculi and Applications '99, L'Aquila, Italy, vol. 1581 of Lecture Notes in Computer Science, pp. 213–227. Springer (1999)
16. Levy, P.B.: λ-calculus and cartesian closed categories. Essay for Part III of the Mathematical Tripos, Cambridge University (1996)
17. Levy, P.B.: Call-by-push-value: a subsuming paradigm (extended abstract). In: Girard, J.-Y (ed.), In: Proceedings, Typed Lambda-Calculi and Applications, L'Aquila, Italy, vol. 1581 of LNCS, pp. 228–242. Springer (1999)
18. Levy, P.B.: Possible world semantics for general storage in call-by-value. In: Bradfield, J. (ed.), In: Proceedings, 16th Annual Conference of the European Assocation for Computer Science Logic (CSL), vol. 2471 of LNCS, pp. 232–246. Springer (2002)
19. Levy, P.B.: Adjunction models for call-by-push-value with stacks. In: Blute, R., Selinger, P. (eds.), In: Proceedings, 9th Conference on Category Theory and Computer Science, Ottawa, 2002, vol. 69 of Electronic Notes in Theoretical Computer Science (2003)
20. Levy, P.B.: Call-By-Push-Value. A Functional/Imperative Synthesis. Semantic Structures in Computation. Springer (2004)
21. Levy, P.B.: Adjunction models for call-by-push-value with stacks. Theory and Applications of Categories, **14**, 75–110 (2005)
22. Levy, P.B.: Jumbo λ-calculus. In: Proceedings, 33rd International Colloquium on Automata, Languages and Programming, vol. 4052 of LNCS pp. 444–455. Springer (2006)
23. Marz, M.: A fully abstract model for sequential computation. Technical Report CSR-98-6, University of Birmingham, School of Computer Science (1998)
24. Marz, M.: A Fully Abstract Model for Sequential Computation. PhD thesis, Technische Universität Darmstadt, published by Logos-Verlag, Berlin (2000)
25. Moggi, E.: Computational lambda-calculus and monads. In: Proceedings, 4th Annual Symposium on Logic in Computer Science, Pacific Grove, California, pp. 14–23. IEEE (1989)
26. Moggi, E.: Notions of computation and monads. Information and Computation **93**, 55–92 (1991)
27. O'Hearn, P.W.: Opaque types in algol-like languages. Manuscript (1993)
28. Ong, C.H.L.: The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming. PhD thesis, Imperial College of Science and Technology (1988)
29. Plotkin, G.D.: Call-by-name, call-by-value and the λ-calculus. Theoretical Computer Science **1**(1), 125–159 (1975)
30. Plotkin, G.D.: LCF considered as a programming language. Theoretical Computer Science **5**, 223–255 (1977)
31. Pitts, A.M., Stark, I.D.B.: Operational reasoning for functions with local state. In: Gordon, A.D., Pitts, A.M. (eds.), Higher Order Operational Techniques in Semantics, Publications of the Newton Institute, pp. 227–273. Cambridge University Press (1998)

32. Selinger, P.: Control categories and duality: On the categorical semantics of the $\lambda\mu$-calculus. Mathematical Structures in Computer Science **11**(2), 207–260 (2001)
33. Streicher, Th., Reus, B.: Classical logic, continuation semantics and abstract machines. Journal of Functional Programming **8**(6), 543–572 (1998)
34. Tait, W.W.: Intensional interpretation of functionals of finite type I. Journal of Symbolic Logic **32**(2), 198–212 (1967)
35. Winskel, G.: Formal Semantics of Programming Languages. MIT Press (1993)