Call-By-Push-Value: A Subsuming Paradigm (extended abstract)

Paul Blain Levy*

Department of Computer Science, Queen Mary and Westfield College
LONDON E1 4NS pbl@dcs.qmw.ac.uk

Abstract. Call-by-push-value is a new paradigm that subsumes the call-by-name and call-by-value paradigms, in the following sense: both operational and denotational semantics for those paradigms can be seen as arising, via translations that we will provide, from similar semantics for call-by-push-value.
To explain call-by-push-value, we first discuss general operational ideas, especially the distinction between values and computations, using the principle that "a value is, a computation does". Using an example program, we see that the lambda-calculus primitives can be understood as push/pop commands for an operand-stack.
We provide operational and denotational semantics for a range of computational effects and show their agreement. We hence obtain semantics for call-by-name and call-by-value, of which some are familiar, some are new and some were known but previously appeared mysterious.

1 Introduction

1.1 A Single Paradigm

In a recent invited lecture [Rey98], Reynolds, surveying over 30 years of programming language development, called for a common framework for typed call-by-name (CBN) and typed call-by-value (CBV). We consider this an important problem, as the existence of two separate paradigms is troubling:

- it makes each language appear arbitrary (whereas a unified language might be more canonical);
- on a more practical level, each time we create a new style of semantics, e.g. Scott semantics, operational semantics, game semantics, continuation semantics etc., we always need to do it twice—once for each paradigm.

We propose call-by-push-value (CBPV), a new typed paradigm based on Filinski's variant of Moggi's computational λ-calculus [Fil96,Mog91], as a solution to this problem. We will introduce a CBPV language, and give translations from CBN and CBV languages into it. We claim that, via these translations, CBPV "subsumes" CBN and CBV.

* supported by EPSRC research studentship no. 96308344

But what does it mean for one language to subsume another? After all, there are sound, adequate translations from CBN and CBV languages into each other [Plo76,HD97] and into other languages such as linear λ-calculus, Moggi's calculus [BW96,Mog91] and others [Mar98,MC88,JLST98,SJ98]. So we must explain in what sense our translations into CBPV go beyond these "classic" transforms, and why, consequently, CBPV is a solution to Reynolds' problem.

We therefore introduce the following informal criterion. A translation α from language L' into language L is *subsumptive* if every "naturally arising" denotational semantics, operational semantics or equation for L' arises, via α, from a "similar" denotational semantics, operational semantics or equation for L.

The importance of such a translation is that the semanticist need no longer attend to L', because its primitives can be seen as no more than syntactic sugar for complex constructs of L. We shall see in Sect. 1.2 that the classic translations mentioned above are not subsumptive.

The essence of Reynolds' problem can now be expressed as follows:

> Give subsumptive translations from CBN and CBV languages into a single language.

The key features of CBPV that enable it to solve this problem are that

1. it divides Moggi's type constructor T into two type constructors U and F, that give types of *thunks* and of *producers* respectively;
2. it distinguishes between *values* and *computations*;
3. writing V`M for "M applied to V", the λ-calculus primitives can be understood as commands for an *operand-stack*:
   - V` can be read as "push V";
   - $\lambda\mathtt{x}$ can be read as "pop $\mathtt{x}$".

(1) is reminiscent of the division of a monad into an adjunction. However, while an adjunction (with extra structure) gives rise to a model for CBPV, different (non-equivalent) adjunctions can give rise to the same model, because not all of the adjunctional structure is used. This is explained in [Lev98].

Feature (2) is shared with CBV, and feature (3) with CBN. (Indeed the push/pop reading is widely used in implementation of lazy languages [Jon92].)

That our translations into CBPV are subsumptive is too informal a claim to prove, but we have a diverse collection of examples to corroborate it:

- We can give operational semantics for CBPV in big-step, small-step or machine form, and recover standard operational semantics for CBV and CBN. These can be formulated to include various computational effects.
- We can give Scott semantics for CBPV, and recover those for CBN [Plo77] and for CBV [Plo85].
- We can give state-passing semantics for CBPV, and recover the mysterious CBN semantics of O'Hearn [O'H93], and a straightforward CBV semantics.
- We can give continuation semantics for CBPV, and recover the CBV semantics of [Plo76] and the CBN semantics of [SR96] (NB *not* that of [Plo76] which is not quite CBN, as it does not validate the η-law).

- We can give game semantics for CBPV, and recover the CBN game semantics of [HO94] and the CBV game semantics of [AM98].
- We can give an equational theory for CBPV. The equations that this gives us for CBN include the β- and η-laws for functions, which generally fail in CBV. The equations that we obtain for CBV include for example

$$\Gamma, \mathtt{x} : \mathtt{bool} \vdash M = \mathtt{if\ x\ then}\ M[\mathtt{true/x}]\ \mathtt{else}\ M[\mathtt{false/x}] \qquad (1)$$

  which generally fails in CBN. (1) is in fact a special case of the η-law for sum types.
- We can give a (rather messy) categorical semantics for CBPV. From a CBPV-structure we can construct for CBN a cartesian closed category, and for CBV a premonoidal category in the sense of [PR97].
- If we add sum types to both CBN and CBV languages, our translations into CBPV can be extended to include them. While both operational and denotational semantics for sum types differ between CBN and CBV, all the differences are recovered from their translation into CBPV.

After discussing related work, we give an operational account of the principles of CBPV. We add divergence and recursion to the basic language, and provide Scott semantics, which helps to motivate our translations from CBN and CBV into CBPV. Finally, we provide operational and denotational semantics for a range of computational effects.

Acknowledgements I am grateful to M. Fiore, M. Marz, E. Moggi, P. O'Hearn, S. Peyton Jones, J. Power, U. Reddy, J. Reynolds, E. Robinson, H. Thielecke, referees and others for their helpful comments on this and related material.

1.2 Related Work

We briefly give some ways in which other proposed translations from CBN and CBV, even those on which ours are based, are not subsumptive. Of course, the objectives that they were designed to achieve are different.

We first look at cases in which semantics for the source language does not, so far as we can see, extend along the translation.

- It is not evident how to provide operational semantics for the monadic target languages of [BW96,Fil96,Mog91] so as to recover standard operational semantics for the source languages.
- The monad language of [BW96,Mog91] does not provide semantics for CBN, because the translation from CBN into it—like the *thunking transform* from CBN to CBV [HD97,SJ98]—does not preserve the η-law for functions.
- As remarked in [BW96], the linear language used there assumes "commutativity" of effects, so that continuation models, for example, do not arise from it; likewise for the language of [Mar98,MC88].
- The CPS transforms of [SR96,Plo76] do not, of course, preserve Scott semantics.

More subtly, there are cases where a semantics for the source language does extend to one for the target, but not (as subsumptiveness requires) to a "similar" one—the semantics of type becomes more complicated.

- To decompose the CBV predomain[1] model of [Plo85] using $A \to_{\mathsf{CBV}} B = A \to TB$ [BW96,Mog91], we must drop the countable-base condition on predomains, because the *total function space* operation does not preserve it. For example, $\mathbb{N} \to \mathbb{N}$ is a flat, uncountable "predomain".
- The CBN game model of [HO94] can exhibit a linear decomposition $A \to_{\mathsf{CBN}} B = !A \multimap B$ [BW96,Gir87], but types must then denote *games* rather than *arenas*. (Some further problems with this linear approach are discussed in [McC96], and it is abandoned for technical reasons in [AHM98].)

2 Call-by-push-value

We introduce CBPV in this section using an operational account, because (as for CBN and CBV) the operational ideas remain essentially constant across different effects, whereas the range of models is wide.

2.1 Operational Principles and Types

In CBPV we distinguish between *computations* and *values*. Intuitively speaking, a computation *does*, while a value *is*. CBPV has two disjoint classes of types: a computation has a *computation type*, while a value has a *value type*. For clarity we underline computation types.

The two classes of types are given by

$$\begin{array}{rcl} A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \\ \underline{B} & ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \end{array} \tag{2}$$

where each set I of *tags* is countable (so the language is infinitary).

We explain the types as follows; notice how this explanation maintains the does/is principle. Throughout execution, there is an *operand-stack* of values and tags that is pushed onto and popped from.

- A value of type $U\underline{B}$ is a *thunk* of a computation of type $\underline{B}$.
- A value of type $\sum_{i\in I} A_i$ is a pair (i, V), where $i \in I$ and V is a value of type A_i.
- A value of type $A \times A'$ is a pair (V, V'), where V is a value of type A and V' is a value of type A'.
- A value of type 1 is the 0-tuple $()$. We largely omit further mention of this type, as it is entirely analogous to $\times$.

[1] A *predomain* $(X, \leqslant)$ is a countably based, algebraic directed-complete poset, with joins of all nonempty bounded subsets, in which the down-set $\{y \in X : y \leqslant x\}$ of each $x \in X$ has a least element. (The last condition is adapted from [AM98]). A *domain* $(X, \leqslant, \bot)$ is a predomain with a least element $\bot$.

- A computation of type FA *produces* a value of type A.
- A computation of type $\prod_{i\in I}\underline{B}_i$ *pops* a tag $i \in I$ from the operand-stack, 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 from the operand-stack, and then behaves as a computation of type $\underline{B}$.

A computation can perform other effects besides popping and producing. For example, a computation M of type $A \to FA'$ might output, then pop a value of type A, then push a value of type C, then input, then pop a value of type C and finally produce a value of type A'. Or it might crash, diverge, make some choices, jump out etc. But it cannot perform any further effects after producing, for then another computation begins, using the value that M produced.

Values alone can be stored, input, output, pushed, popped or chosen. Identifiers can be bound to (or replaced by) values alone, and therefore they always have value type. A computation is too "active" for this, although a *thunk* of a computation M is a value, so it can be stored etc. Later the thunk can be *forced*, and M then happens. Of course, a single thunk can be forced several times.

We call a value type of the form $\sum_{i\in I}1$ a *groundtype* and write $\underline{n}$ or even just n for $(n, ())$. In particular, we write `bool` and `nat` for the groundtypes $\sum_{i\in\{\mathsf{true},\mathsf{false}\}}1$ and $\sum_{i\in\mathbb{N}}1$ respectively. A computation of type $F\sum_{i\in I}1$ is called a *ground producer* because it produces a ground value.

Moggi's type TA [Mog91] becomes in our type system UFA, because a value of type TA is a thunk of a computation that produces a value of type A.

2.2 The basic language

Definition 1. A *context* Γ is a finite sequence of identifiers with value types $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{m-1} : A_{m-1}$. Sometimes we omit the identifiers and write Γ as a list of value types.

The calculus has two kinds of judgement

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

for computations and values respectively. The terms are defined by Fig. 1. We include `let`, although it could be regarded as sugar. Note that $\prod$ is a projection product, whereas $\times$ is a pattern-match product. The key computations are

- `produce` V, the trivial producer of V;
- M `to x in` M', the sequenced computation (called "generalized `let`" by Filinski [Fil96]) where firstly the producer M happens, and if it produces a value V then M' happens with `x` bound to V.

Imperatively, V‘ means "push V" and $\lambda\mathtt{x}$ means "pop `x`". This reading is illustrated in Sect. 2.3.

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

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

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

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

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

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

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

$\mathtt{pm}$ is an abbreviation for $\mathtt{pattern} - \mathtt{match}$.

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

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

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

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

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

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

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

$$\frac{}{\lambda \mathtt{x} M \Downarrow \lambda \mathtt{x} M}$$

Fig. 1. Terms of Basic Language, and Big-Step Semantics

Remark 1. The reader may wonder why we have not included *complex values* such as $\mathtt{x} : A \times A' \vdash^{\mathrm{v}} \mathtt{pm\ x\ as\ (y, z)\ in\ y} : A$ or arithmetic expressions. The reason is that they somewhat complicate the operational semantics, our presentation of which exploits the fact that values do not need to be evaluated. Consequently, and since they lie outside the range of our translations from CBN and CBV, we omit them, except in the example program of Sect. 2.3. Nonetheless, all our denotational and categorical models can interpret them straightforwardly.

2.3 Example Computation

The following example M illustrates the naive imperative reading of CBPV. To this end, we add to the language arithmetic expressions as values (Remark 1) and the facility to prefix a `print` command to any computation.

```
print "hello0";
let x be 3 in
let y be thunk (
            print "hello1";
            λz
            print "we just popped "z;
            produce x + z
         ) in
print "hello2";
( print "hello3";
  7‘
  print "we just pushed 7";
  force y
) to w in
print "w is bound to "w;
produce w + 5
```

Note that if the word `thunk` were omitted, M would be ill-typed, because `y` can identify only a value, not a computation. The type of `y` is $U(\mathtt{nat} \to F\,\mathtt{nat})$, because `y` identifies a thunk of a computation that pops a natural number and then produces a natural number.

M outputs as follows

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

and finally produces the value 15.

It is clear that if the lines `print "hello1"` and λ`z` were exchanged, or if the lines `print "hello3"` and 7‘ were exchanged, the behaviour of M would be unchanged. We say that "effects commute with λ and with ‘ ". A more familiar example of this phenomenon is the equivalence of λ`x diverge` and `diverge`. (We are assuming here that, as in our example, the global computation is a producer, so there is no danger that we will try to pop from an empty stack.)

2.4 Big-Step Operational Semantics

Terminal computations (a subset of closed computations) are given by

$$T ::= \quad \texttt{produce}\, V \;\mid\; \lambda\langle \ldots, i.M_i, \ldots\rangle \;\mid\; \lambda \mathtt{x} M \tag{3}$$

Intuitively these are computations that cannot proceed if the operand-stack is empty. We write $\mathbb{C}_{\underline{B}}$ for the set of closed computations of type $\underline{B}$, $\mathbb{T}_{\underline{B}}$ for the set of terminal elements of $\mathbb{C}_{\underline{B}}$, and $\mathbb{V}_A$ for the set of closed values of type A.

For the basic language, we define in Fig. 1 a relation $\Downarrow$ from $\mathbb{C}_{\underline{B}}$ to $\mathbb{T}_{\underline{B}}$. It can be proved to be a total function. Note that only computations happen; values do not need to be evaluated.

2.5 Equations and Observational Equivalence

We form an equational theory whose axioms are the equations in Fig. 2. Compare this theory to those of CBN and CBV.

- In CBV, equations such as η for $+$ types hold because *an identifier can be bound only to a value.*
- In CBN, equations such as η for $\to$ types hold because *a term of $\to$ type can be evaluated only by applying it.*

Since CBPV has both of these features, it has both kinds of equation, which is essentially why it can subsume both paradigms.

Definition 2. A *ground context* $\mathcal{C}[]$ is a closed ground producer with zero or more occurrences of a hole which can be either a computation or a value.

Definition 3. We say that $M \simeq M'$ when for all ground contexts $C[]$, $\mathcal{C}[M] \Downarrow \texttt{produce}\, \underline{n}$ iff $\mathcal{C}[M'] \Downarrow \texttt{produce}\, \underline{n}$.

In all of our CBPV languages (e.g. in Sect. 5.1) the equations of the theory hold as observational equivalences (for the appropriate variation on Def. 3). As usual, this will follow from the soundness and adequacy of our models.

It is worth noticing that, with our imperative understanding of V‘ and λ`x`, the β-law for $\to$ equates "push V, then pop `x`, then M" with $M[V/\mathtt{x}]$. Similarly, the η-law for $\to$ equates M (in which `x` is not free) with "pop `x`, then push `x`, then M". These are both intuitively compelling.

$$
\begin{array}{lll}
\Gamma \vdash_c \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V\ \mathtt{in}\ M & = M[V/\mathtt{x}] & : \underline{B} \\
\Gamma \vdash_c (\mathtt{produce}\ V)\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ M & = M[V/\mathtt{x}] & : \underline{B} \\
\Gamma \vdash_c \mathtt{force}\ \mathtt{thunk}\ M & = M & : \underline{B} \\
\Gamma \vdash_c \mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \ldots, (i, \mathtt{x})\ \mathtt{in}\ M_i, \ldots & = M_{\hat{\imath}}[V/\mathtt{x}] & : \underline{B} \\
\Gamma \vdash_c \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y})\ \mathtt{in}\ M & = M[V/\mathtt{x}, V'/\mathtt{y}] & : \underline{B} \\
\Gamma \vdash_c \hat{\imath}\text{`}\lambda\langle \ldots, i.M_i, \ldots\rangle & = M_{\hat{\imath}} & : \underline{B}_{\hat{\imath}} \\
\Gamma \vdash_c V\text{`}\lambda\mathtt{x}M & = M[V/\mathtt{x}] & : \underline{B}
\end{array}
$$

$$
\begin{array}{llll}
\Gamma \vdash_c M & = M\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ \mathtt{produce}\ \mathtt{x} & : FA & (\mathtt{x} \notin \Gamma) \\
\Gamma \vdash_v V & = \mathtt{thunk}\ \mathtt{force}\ V & : U\underline{B} & \\
\Gamma \vdash_c M[V/\mathtt{z}] & = \mathtt{pm}\ V\ \mathtt{as}\ \ldots, (i, \mathtt{x})\ \mathtt{in}\ M[(i, \mathtt{x})/\mathtt{z}], \ldots & : \underline{B} & (\mathtt{x} \notin \Gamma) \\
\Gamma \vdash_c M[V/\mathtt{z}] & = \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y})\ \mathtt{in}\ M[(\mathtt{x}, \mathtt{y})/\mathtt{z}] & : \underline{B} & (\mathtt{x}, \mathtt{y} \notin \Gamma) \\
\Gamma \vdash_c M & = \lambda\langle \ldots, i.i\text{`}M, \ldots\rangle & : \prod_{i \in I} \underline{B}_i & \\
\Gamma \vdash_c M & = \lambda\mathtt{x}\ \mathtt{x}\text{`}M & : A \to \underline{B} & (\mathtt{x} \notin \Gamma)
\end{array}
$$

$$
\Gamma \vdash_c (M\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ M')\ \mathtt{to}\ \mathtt{y}\ \mathtt{in}\ M'' = M\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ (M'\ \mathtt{to}\ \mathtt{y}\ \mathtt{in}\ M'') : \underline{B}\ \ (\mathtt{x}, \mathtt{y} \notin \Gamma)
$$

$$
\begin{array}{llll}
\Gamma \vdash_c & \hat{\imath}\text{`}(M\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ M') & = & M\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ \hat{\imath}\text{`}M' \quad : \underline{B}_{\hat{\imath}}\ (\mathtt{x} \notin \Gamma) \\
\Gamma \vdash_c & V\text{`}(M\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ M') & = & M\ \mathtt{to}\ \mathtt{x}\ \mathtt{in}\ V\text{`}M' \quad : \underline{B}\ (\mathtt{x} \notin \Gamma)
\end{array}
$$

Fig. 2. β-laws, η-laws and other laws

3 Divergence, Recursion and Scott Semantics

As divergence is the computational effect most familiar to semanticists, we study it first. We add to the basic language the computations

$$
\frac{}{\Gamma \vdash_c \mathtt{diverge} : \underline{B}} \qquad \frac{\Gamma, \mathtt{x} : U\underline{B} \vdash_c M : \underline{B}}{\Gamma \vdash_c \mu\mathtt{x}M : \underline{B}}
$$

and the big-step rules

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

so that $\Downarrow$ is now a partial function from $\mathbb{C}_{\underline{B}}$ to $\mathbb{T}_{\underline{B}}$. The recursion binder $\mu\mathtt{x}$ can be read imperatively as "bind-to-a-thunk-of-the-present-computation $\mathtt{x}$", and therefore $\mu\mathtt{x}M$ is a computation.

The Scott semantics for CBPV interprets value types (and hence contexts) as predomains and computation types as domains. For example,

- $[\![FA]\!]$ is the lift of $[\![A]\!]$;
- if $[\![\underline{B}]\!]$ is the domain $(X, \leqslant, \bot)$ then $[\![U\underline{B}]\!]$ is its underlying predomain $(X, \leqslant)$;
- $[\![A \to \underline{B}]\!]$ is the domain of continuous functions from $[\![A]\!]$ to $[\![\underline{B}]\!]$

Then to each computation $\Gamma \vdash_c M : \underline{B}$ we associate a continuous function $[\![M]\!] : [\![\Gamma]\!] \to [\![\underline{B}]\!]$, and to each value $\Gamma \vdash_v V : A$ we associate a continuous function $[\![V]\!] : [\![\Gamma]\!] \to [\![A]\!]$. For example, where $\rho \in [\![\Gamma]\!]$,

$$
[\![\mathtt{produce}\ V]\!]\rho = \mathsf{lift}\ ([\![V]\!]\rho)
$$

$$[\![M \texttt{ to x in } N]\!]\rho = \begin{cases} \bot & \text{if } [\![M]\!]\rho = \bot \\ [\![N]\!](\rho, \mathtt{x} \mapsto x) & \text{if } [\![M]\!]\rho = \mathsf{lift}\ x \end{cases}$$
$$[\![\texttt{thunk } M]\!]\rho = [\![M]\!]\rho$$
$$[\![\texttt{force } V]\!]\rho = [\![V]\!]\rho$$

In particular, $[\![\texttt{thunk diverge}]\!]\rho$ is the least element of the predomain $[\![U\underline{B}]\!]$.

Proposition 1 (Soundness/Adequacy). For any closed computation M,

1. if $M \Downarrow T$, then $[\![M]\!] = [\![T]\!]$;
2. if $[\![M]\!] > \bot$, then $M \Downarrow T$ for some T.

4 Translating CBN and CBV into CBPV

As we would expect from the Scott semantics of Sect. 3, CBN types translate into computation types, while CBV types translate into value types. The most important type decomposition into CBPV is

$$\underline{B} \to_{\mathsf{CBN}} \underline{B}' = (U\underline{B}) \to \underline{B}' \tag{4}$$

This corresponds to the fact that in CBN a function is effectively applied to a thunk. Perhaps it is because the interpretation of U and of `thunk` is almost invisible in CBPV Scott semantics that this decomposition has remained hidden for so long.

Another important type decomposition into CBPV is

$$A \to_{\mathsf{CBV}} A' = U(A \to FA') \tag{5}$$

This is similar, and in a sense equivalent, to Moggi's decomposition [Mog91] as $A \to TB$, but notice that (5) avoids the countability problem mentioned in Sect. 1.1. It says that a CBV function from A to A' is a thunk of a computation that pops a value of type A and then produces a value of type A'.

The translations into CBPV are given in Fig. 3 and Fig. 4. The source languages of these translations are prototypical CBN and CBV languages like PCF and $\mathrm{PCF}_{\mathsf{v}}$, with sum types. They are equipped with Scott semantics $[\![-]\!]_{\mathsf{CBN}}$ and $[\![-]\!]_{\mathsf{CBV}}$ (together with a semantics $[\![-]\!]^{\mathsf{val}}_{\mathsf{CBV}}$ for CBV values) and big-step semantics $\Downarrow_{\mathsf{CBN}}$ and $\Downarrow_{\mathsf{CBV}}$. We omit presenting them in detail. For simplicity, we have supplied a projection product for CBN but a pattern-match product for CBV; although in principle one could have both kinds of product in each paradigm.

Some of the technical results for the CBN translation concern not the *function* $-^{\mathsf{n}}$ (which does not commute with substitution) but a *relation* $\mapsto^{\mathsf{n}}$ from CBN to CBPV terms. Informally, $M \mapsto^{\mathsf{n}} M'$ means that M' is M^{n} with possibly some extra `force thunk` prefixes. The direct inductive definition of $\mapsto^{\mathsf{n}}$ is comprised of one rule for each CBN term-constructor, e.g.

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

C	C^{n} (a computation type)
$\mathtt{bool}$	$F\sum_{b\in\{\mathsf{true},\mathsf{false}\}}1$
$A\to B$	$UA^{\mathsf{n}}\to B^{\mathsf{n}}$
$A\times B$	$A^{\mathsf{n}}\ \pi\ B^{\mathsf{n}}$
$A+B$	$F(UA^{\mathsf{n}}+UB^{\mathsf{n}})$

$A_0,\ldots,A_{m-1}\vdash M:C$	$UA_0{}^{\mathsf{n}},\ldots,UA_{m-1}{}^{\mathsf{n}}\vdash^{\mathsf{c}} M^{\mathsf{n}}:C^{\mathsf{n}}$
$\mathtt{x}$	$\mathtt{force\ x}$
$\mathtt{false}$	$\mathtt{produce}\ \underline{\mathsf{false}}$
$\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N'$	$M^{\mathsf{n}}\ \mathtt{to\ z\ in\ pm\ z\ as}\ \underline{\mathsf{true}}\ \mathtt{in}\ N^{\mathsf{n}},\underline{\mathsf{false}}\ \mathtt{in}\ N'^{\mathsf{n}}$
$\lambda\mathtt{x}M$	$\lambda\mathtt{x}M^{\mathsf{n}}$
$N\text{`}M$	$(\mathtt{thunk}\ N^{\mathsf{n}})\text{`}M^{\mathsf{n}}$
$\langle M,M'\rangle$	$\lambda\langle 0.M^{\mathsf{n}},1.M'^{\mathsf{n}}\rangle$
πM	$0\text{`}M^{\mathsf{n}}$
$\mathtt{inl}\ M$	$\mathtt{produce\ inl\ thunk}\ M^{\mathsf{n}}$
$\mathtt{pm}\ M\ \mathtt{as\ inl\ x\ in}\ N,\mathtt{inr\ x\ in}\ N'$	$M\ \mathtt{to\ z\ in\ pm\ z\ as\ inl\ x\ in}\ N^{\mathsf{n}},\mathtt{inr\ x\ in}\ N'^{\mathsf{n}}$
$\mu\mathtt{x}M$	$\mu\mathtt{x}M^{\mathsf{n}}$

Fig. 3. Translation of CBN types and terms

C	C^{v} (a value type)
$\mathtt{bool}$	$\sum_{b\in\{\mathsf{true},\mathsf{false}\}}1$
$A\to B$	$U(A^{\mathsf{v}}\to FB^{\mathsf{v}})$
$A\times B$	$A^{\mathsf{v}}\times B^{\mathsf{v}}$
$A+B$	$A^{\mathsf{v}}+B^{\mathsf{v}}$

$A_0,\ldots,A_{m-1}\vdash V:C$	${A_0}^{\mathsf{v}},\ldots,{A_{m-1}}^{\mathsf{v}}\vdash^{\mathsf{v}} V^{\mathsf{val}}:C^{\mathsf{v}}$
$\mathtt{x}$	$\mathtt{x}$
$\mathtt{false}$	$\underline{\mathsf{false}}$
$\lambda\mathtt{x}M$	$\mathtt{thunk}\ \lambda\mathtt{x}M^{\mathsf{v}}$
$\mu\mathtt{y}\lambda\mathtt{x}M$	$\mathtt{thunk}\ \mu\mathtt{y}\lambda\mathtt{x}M^{\mathsf{v}}$
(V,V')	$(V^{\mathsf{val}},V'^{\mathsf{val}})$
$\mathtt{inl}\ V$	$\mathtt{inl}\ V^{\mathsf{val}}$

$A_0,\ldots,A_{m-1}\vdash M:C$	${A_0}^{\mathsf{v}},\ldots,{A_{m-1}}^{\mathsf{v}}\vdash^{\mathsf{c}} M^{\mathsf{v}}:FC^{\mathsf{v}}$
V (a value)	$\mathtt{produce}\ V^{\mathsf{val}}$
$\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N'$	$M^{\mathsf{v}}\ \mathtt{to\ z\ in\ pm\ z\ as}\ \underline{\mathsf{true}}\ \mathtt{in}\ N^{\mathsf{n}},\underline{\mathsf{false}}\ \mathtt{in}\ N'^{\mathsf{n}}$
MN (M first)	$M^{\mathsf{v}}\ \mathtt{to\ f\ in}\ N^{\mathsf{v}}\ \mathtt{to\ x\ in\ x`(force\ f)}$
$\mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y})\ \mathtt{in}\ N$	$M^{\mathsf{v}}\ \mathtt{to\ z\ in\ pm\ z\ as}\ (\mathtt{x},\mathtt{y})\ \mathtt{in}\ N'^{\mathsf{v}}$
$\mathtt{pm}\ M\ \mathtt{as\ inl\ x\ in}\ N,\mathtt{inr\ x\ in}\ N'$	$M^{\mathsf{v}}\ \mathtt{to\ z\ in\ pm\ z\ as\ inl\ x\ in}\ N^{\mathsf{v}},\mathtt{inr\ y\ in}\ N'^{\mathsf{v}}$

Fig. 4. Translation of CBV types, values and terms

and the additional rule $$\frac{M \mapsto^{\mathsf{n}} M'}{M \mapsto^{\mathsf{n}} \mathtt{force}\ \mathtt{thunk}\ M'}$$

Proposition 2. 1. $(M[V/\mathtt{x}])^{\mathsf{v}} = M^{\mathsf{v}}[V^{\mathsf{val}}/\mathtt{x}]$
2. If $M \mapsto^{\mathsf{n}} M'$ and $N \mapsto^{\mathsf{n}} N'$ then $M[N/\mathtt{x}] \mapsto^{\mathsf{n}} M'[\mathtt{thunk}\ N'/\mathtt{x}]$

We are now in a position to describe the fundamental subsumption properties: that the Scott and big-step semantics of CBN and CBV can be recovered from those of CBPV.

The preservation of the Scott semantics is straightforward:

Proposition 3. 1. If A is a CBN type then $[\![A]\!]_{\mathsf{CBN}} = [\![A^{\mathsf{n}}]\!]$
2. If $\Gamma \vdash M : A$ is a CBN term and $M \mapsto^{\mathsf{n}} M'$ then $[\![M]\!]_{\mathsf{CBN}} = [\![M']\!]$
3. If A is a CBV type then $[\![A]\!]_{\mathsf{CBV}} = [\![A^{\mathsf{v}}]\!]$
4. If $\Gamma \vdash V : A$ is a CBV value then $[\![V]\!]^{\mathsf{val}}_{\mathsf{CBV}} = [\![V^{\mathsf{val}}]\!]$
5. If $\Gamma \vdash M : A$ is a CBV term then $[\![M]\!]_{\mathsf{CBV}} = [\![M^{\mathsf{v}}]\!]$

That the equations of CBN/CBV are preserved follows from Prop. 2.

Proposition 4. Suppose M is a closed CBN term, and $M \mapsto^{\mathsf{n}} M'$.

1. If M' is terminal then M is, and M is terminal iff M^{n} is.
2. If $M \Downarrow_{\mathsf{CBN}} T$ then, for some T', $T \mapsto^{\mathsf{n}} T'$ and $M' \Downarrow T'$.
3. If $M' \Downarrow T'$, then, for some T, $T \mapsto^{\mathsf{n}} T'$ and $M \Downarrow_{\mathsf{CBN}} T$.

Proposition 5. Suppose M is a closed CBV term.

1. M is terminal iff M^{v} is terminal.
2. If $M \Downarrow_{\mathsf{CBV}} T$ then $M^{\mathsf{v}} \Downarrow T^{\mathsf{v}}$.
3. If $M^{\mathsf{v}} \Downarrow T'$, then, for some T, $T^{\mathsf{v}} = T'$, and $M \Downarrow_{\mathsf{CBV}} T$.

Parts (2) and (3) of these are proved by induction, primarily on the big-step derivation, and secondarily on $\mapsto^{\mathsf{n}}$ (for Prop. 4) or M (for Prop. 5).

5 Operational Semantics for Computational Effects

It is straightforward to adapt the big-step semantics of Sect. 2.4 to various computational effects (except for control effects, which require *machine semantics*, where the search for a redex is made explicit). We give two examples: global store and nondeterminism.

5.1 Global Groundtype Store

We will consider a single global storage cell $\mathtt{X}$ that stores a value of groundtype $\sum_{s\in S} 1$. We add to the basic langugage the computations

$$\frac{}{\Gamma \vdash^{\mathsf{c}} \mathtt{deref}\ \mathtt{X} : F\textstyle\sum_{s\in S} 1} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{s\in S} 1 \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} X := V; M : \underline{B}}$$

While it is possible to give type $F1$ to commands such as assignment and output, here we regard them as prefixes.

We define a relation $\Downarrow$ from $S \times \mathbb{C}_{\underline{B}}$ to $S \times \mathbb{T}_{\underline{B}}$, adapting the rules of Sect. 2.4 and adding rules for the new constructs. For example:

$$\frac{}{s, T \Downarrow s, T} \qquad \frac{s, M \Downarrow s', \lambda \mathtt{x} N \quad s', N[V/\mathtt{x}] \Downarrow s'', T}{s, V \text{`} M \Downarrow s'', T}$$

$$\frac{}{s, \mathtt{deref}\ \mathtt{X} \Downarrow s, \mathtt{produce}\ \underline{s}} \qquad \frac{s', M \Downarrow s'', T}{s, \mathtt{X} := \underline{s'}; M \Downarrow s'', T}$$

$\Downarrow$ can be proved to be a total function.

Finally, we say that $M \simeq M'$ when for all ground contexts $\mathcal{C}[]$ and $s, s' \in S$, $s, \mathcal{C}[M] \Downarrow s', \mathtt{produce}\ \underline{n}$ iff $s, \mathcal{C}[M'] \Downarrow s', \mathtt{produce}\ \underline{n}$.

5.2 Nondeterminism

We add to the basic language the divergence and recursion facilities of Sect. 3 together with the following term and big-step rule:

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{choose}\ \mathtt{x}\ M : \underline{B}} \qquad \frac{M[V/\mathtt{x}] \Downarrow T}{\mathtt{choose}\ \mathtt{x}\ M \Downarrow T}$$

6 Denotational Semantics for Computational Effects

We describe denotational semantics for the effects of Sect. 5. Part is easy: a value type (or a context) should denote a set, with $\times$ and $\sum$ interpreted in the usual way, and a value $\Gamma \vdash^{\mathsf{v}} V : A$ should denote a function $[\![V]\!] : [\![\Gamma]\!] \longrightarrow [\![A]\!]$.

The remainder of the semantics differs between the effects. While logically we should present the various semantics first, and then state the soundness results, this makes the interpretation of type constructors appear ad hoc. So we will proceed in reverse order. For global store and nondeterminism, we will state first the soundness and adequacy theorems that we are aiming to achieve, even though they are not yet meaningful, and use this to motivate the semantics. We will also give continuation semantics for the basic language. (Using machine semantics, this can be similarly motivated.)

Proposition 6 (Soundness/Adequacy). Let M be a closed computation.

1. For global store, if $s, M \Downarrow s', T$ then $[\![M]\!]s = [\![T]\!]s'$.
2. For nondeterminism, $[\![M]\!] = \bigcup_{M \Downarrow T} [\![T]\!]$.

By looking at Prop. 6, we can guess the interpretation of a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$. (Recall that if $\underline{B} = FA$ then this judgement corresponds to a CBV term of type A, so its interpretation is familiar.)

- For global store, $[\![M]\!]$ will be a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$, where $[\![\underline{B}]\!]$ is a set. If $\underline{B} = FA$ then $[\![\underline{B}]\!] = S \times [\![A]\!]$, so that $[\![M]\!]$ is a function from $S \times [\![\Gamma]\!]$ to $S \times [\![A]\!]$.
- For nondeterminism, $[\![M]\!]$ will be a relation from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$, where $[\![\underline{B}]\!]$ is a set. If $\underline{B} = FA$, then $[\![\underline{B}]\!] = [\![A]\!]$, so that $[\![M]\!]$ is a relation from $[\![\Gamma]\!]$ to $[\![A]\!]$.
- For continuation semantics, $[\![M]\!]$ will be a function from $[\![\Gamma]\!] \times [\![\underline{B}]\!]$ to Ans (a fixed set that we regard as the set of "answers"), where $[\![\underline{B}]\!]$ is a set. If $\underline{B} = FA$, then $[\![\underline{B}]\!] = [\![A]\!] \to \mathsf{Ans}$, so that $[\![M]\!]$ is a function from $[\![\Gamma]\!] \times ([\![A]\!] \to \mathsf{Ans})$ to Ans.

We next turn our attention to the interpretation of U, $\prod_{i \in I}$ and $\to$. For U, we know that values $\Gamma \vdash^{\mathsf{v}} U\underline{B}$ correspond to computations $\Gamma \vdash^{\mathsf{c}} \underline{B}$. Thus, in the case of global store, functions from $[\![\Gamma]\!]$ to $[\![U\underline{B}]\!]$ must correspond to functions from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$. Therefore we set $[\![U\underline{B}]\!] = S \to [\![\underline{B}]\!]$. Similarly we can determine the interpretation of U for each effect. As expected, it follows in each case that UFA denotes the same set as Moggi's type TA [Mog91]:

effect	U	F	$T = UF$
global store	$S \to -$	$S \times -$	$S \to (S \times -)$
nondeterminism	$\mathcal{P}$	$-$	$\mathcal{P}$
control	$- \to \mathsf{Ans}$	$- \to \mathsf{Ans}$	$(- \to \mathsf{Ans}) \to \mathsf{Ans}$

For $A \to \underline{B}$, we know that computations $\Gamma \vdash^{\mathsf{c}} A \to \underline{B}$ correspond to computations $\Gamma, A \vdash^{\mathsf{c}} \underline{B}$. Thus, in the case of nondeterminism, relations from $[\![\Gamma]\!]$ to $[\![A \to \underline{B}]\!]$ must correspond to relations from $[\![\Gamma]\!] \times [\![A]\!]$ to $[\![\underline{B}]\!]$. Therefore we set $[\![A \to \underline{B}]\!]$ to be $[\![A]\!] \times [\![\underline{B}]\!]$. Similar reasoning suggests interpretations for both $\to$ and $\prod_{i \in I}$ for each of our effects:

effect	$\prod_{i \in I}$	$\to$
global store	$\prod_{i \in I}$	$\to$
nondeterminism	$\sum_{i \in I}$	$\times$
control	$\sum_{i \in I}$	$\times$

We omit the straightforward semantics of terms.

Proposition 7. These five denotational semantics for CBPV all validate the equations of Sect. 2.5. More precisely, if $M = M'$ is provable in the equational theory then $[\![M]\!] = [\![M']\!]$.

Prop. 6 is now meaningful and can be proved. In particular, (1) is trivial.

All these models induce models for CBN and CBV. For CBV we recover the familiar continuation semantics of $A \to_{\mathsf{CBV}} A'$ as $(A \times (A' \to \mathsf{Ans})) \to \mathsf{Ans}$. For CBN we recover the continuation semantics of [SR96], and also, from our CBPV global store semantics, the state-passing semantics of [O'H93].

References

[AHM98] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. Proceedings, Thirteenth Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1998.

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

[BW96] N. Benton and P. Wadler. Linear logic, monads and the lambda calculus. In *Proceedings, 11$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science*, pages 420–431, New Brunswick, 1996. IEEE Computer Society Press.

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

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[HD97] J. Hatcliff and O. Danvy. Thunks and the λ-calculus. *Journal of Functional Programming*, 7(3):303–319, May 1997.

[HO94] M. Hyland and L. Ong. On full abstraction for PCF. submitted, 1994.

[JLST98] S. Peyton Jones, J. Launchbury, M. Shields, and A. Tolmach. Bridging the gulf: A common intermediate language for ML and Haskell. In *Proc. 25th ACM Symposium on Principles of Programming Languages*, San Diego, 1998.

[Jon92] S. L. Peyton Jones. Implementing lazy functional languages on stock hardware: the spineless tagless G-machine. *Journal of Functional Programming*, 2(2):127–202, July 1992.

[Lev98] P. B. Levy. Categorical aspects of call-by-push-value. draft, available at `http://www.dcs.qmw.ac.uk/ pbl/papers.html`, 1998.

[Mar98] M. Marz. A fully abstract model for sequential computation. draft, 1998.

[MC88] A. Meyer and S. Cosmodakis. Semantical Paradigms. In *Proc. Third Annual Symposium on Logic in Computer Science.* Computer Society Press, 1988.

[McC96] G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types.* PhD thesis, University of London, 1996.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[O'H93] P. W. O'Hearn. Opaque types in algol-like languages. manuscript, 1993.

[Plo76] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(1):125–159, 1976.

[Plo77] G. D. Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5, 1977.

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

[PR97] A. J. Power and E. P. Robinson. Premonoidal categories and notions of computation. *Math. Struct. in Comp. Sci.*, 7(5):453–468, October 1997.

[Rey98] J. Reynolds. Where theory and practice meet: POPL past and future. Invited Lecture, 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California, January 19–21, 1998.

[SJ98] M. Shields and S. Peyton Jones. Bridging the gulf better. Draft, 1998.

[SR96] Th. Streicher and B. Reus. Continuation semantics, abstract machines and control operators. submitted to Journal of Functional Programming, 1996.