A Proof of Strong Normalization for Call-by-push-value

Christian Doczkal [a], Jan Schwinghammer [a,*]

[a] *Programming Systems Lab, Saarland University, Saarbrücken, Germany*

Abstract

The call-by-push-value (CBPV) calculus is a general framework within which one can study computational effects such as state, nondeterminism, and input/output. Compared to the simply typed lambda calculus, the CBPV type system is much more fine-grained and distinguishes between value and computation types. We give a self-contained proof of strong normalization for CBPV, employing a notion of $\top\top$-closure to adapt the Girard-Tait method to CBPV computation types.

Key words: Programming calculi, formal semantics, normalization

1 Introduction

The call-by-push-value (CBPV) calculus was introduced by Levy as a means to study the call-by-name and call-by-value semantics of lambda calculus in a single framework [4,5]. As a consequence, CBPV is much more fine-grained than simply typed lambda calculus: a central idea is to distinguish values from computations, both at the level of terms and types.

While there is a great amount of evidence that CBPV achieves the goal of providing a useful meta language for semantics, less is known about its operational properties. Previously, operational aspects of CBPV have only been considered in the form of big-step evaluation, where termination is comparatively easy to establish, and a (deterministic) CK-machine semantics.

In this short note we prove the strong normalization of the calculus, when equipped with full (non-deterministic) $\beta\eta$ reduction.

1.1 The Problem

In the classic Girard-Tait reducibility method for proving normalization, a family of sets of 'reducible' terms is defined by induction on types [11,3]. For instance, a term of function type $A \to B$ is reducible iff its application to reducible arguments (of type A) yields a reducible term (of type B).

Adapting the definition of these predicates to call-by-push-value is problematic. For example, CBPV provides a type FA, of computations that return values of type A. The only way to deconstruct terms M of this type is by sequencing, M **to** $x.N$, where the type $\underline{B}$ of N need not be related to A in any way. As a consequence the definition of the set of reducible terms of type FA cannot in general refer to reducibility of terms of type $\underline{B}$.

Indeed, the same problem arises in Moggi's computational metalanguage [9], and also in the simply typed lambda calculus with sums, where the elimination constructs for monadic and sum types, resp., prevent straightforward inductive definitions of reducibility (*cf.* [10,1]). An alternative to a direct proof of strong normalization is to consider simulations in calculi that are already known to be terminating. E.g. [2] considers encodings of the lambda calculus with sums into the simply typed lambda calculus. While such an approach could be possible for CBPV, we note that already in [2] the encoding is rather subtle and non-trivial.

1.2 Reducibility and Stacks

Inspired by recent work of Lindley and Stark on the strong normalization of Moggi's computational metalanguage, we prove the strong normalization of CBPV *with stacks*. The central proof idea is the $\top\top$-lifting suggested in [7], although we found it slightly more convenient to use a $\top\top$-*closure* operation. With the help of stacks that represent evaluation contexts, one can directly observe strong normalization of arbitrary computations.

In fact, in using $\top\top$-closure we obtain a pleasingly

* Corresponding author.

URLs: www.ps.uni-sb.de/~doczkal (Christian Doczkal), www.ps.uni-sb.de/~jan (Jan Schwinghammer).

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

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_j}{\Gamma \vdash^{\mathsf{v}} \langle j, V\rangle : \sum_{i\in I} A_i}(j \in I) \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i\in I} A_i \qquad \forall i \in I.\ \Gamma, x{:}A_i \vdash^{\mathsf{c}} M_i : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathbf{pm}\ V\ \mathbf{as}\ \{\ldots, \langle i, x\rangle .M_i, \ldots\} : \underline{B}}$$

$$\frac{}{\Gamma \vdash^{\mathsf{v}} \star : \mathbf{1}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} (V, V') : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \qquad \Gamma, x{:}A, y{:}A' \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathbf{pm}\ V\ \mathbf{as}\ (x, y) .M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} \mathbf{return}\, V : FA} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : FA \qquad \Gamma, x{:}A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathbf{to}\ x.N : \underline{B}} \qquad \frac{\forall i \in I.\ \Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} \lambda\{\ldots, i.M_i, \ldots\} : \prod_{i\in I} \underline{B}_i}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \prod_{i\in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} M\ j : \underline{B}_j}(j \in I) \qquad \frac{\Gamma, x{:}A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda x.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : A \to \underline{B} \qquad \Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} M\, V : \underline{B}}$$

Fig. 1. CBPV terms

straightforward, *uniform* definition of reducibility for computations, defined as adjoints to sets of reducible stacks. The strict separation between values and computations in CBPV is emphasized by the fact that reducibility of the former is defined without recourse to elimination constructs.

As in [7], the consideration of stacks allows for a treatment not only of $\beta\eta$ reductions but also of permutation conversions. These reemerge as reductions on stacks, and it is then easily seen that each application decreases the length of the stack.

The next section recalls Levy's calculus (Section 2.1) and presents its small-step operational semantics (Section 2.2). Section 3.1 defines the $\top\top$-closure we have in mind, and the reducibility predicates. Once their properties are developed (Section 3.2), proving strong normalization is a trivial consequence.

2 The Call-by-push-value calculus

2.1 Syntax

2.1.1 Terms and Types

As mentioned above, CBPV distinguishes between *values* denoted by V, W, and *computations*, denoted M, N. Correspondingly there are *value types*, ranged over by A, and *computation types*, ranged over by (underlined) letters $\underline{B}$, $\underline{C}$. Types are defined by the following grammar, where I stands for a finite index set:

$$A ::= U\underline{B} \mid \textstyle\sum_{i\in I} A_i \mid \mathbf{1} \mid A \times A$$
$$\underline{B} ::= FA \mid \textstyle\prod_{i\in I} \underline{B}_i \mid A \to \underline{B}$$

Here, $U\underline{B}$ is the type of 'thunks' of computations of type $\underline{B}$, and $\mathbf{1}$ is a unit type. Note that sums and binary products are value types while general products $\prod_{i\in I} \underline{B}_i$ and function types $A \to \underline{B}$ belong to computation types: $\prod_{i\in I} \underline{B}_i$ has projections, whereas binary products $A \times A'$ (as well as sums) use pattern matching in their respective elimination constructs. Finally, FA is the type of 'returners', that embed values as trivial computations.

Figure 1 defines the (typed) syntax of CBPV: there are judgements $\Gamma \vdash^{\mathsf{v}} V : A$ for values, and $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ for computations. In either case, only value types appear in the type context $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$. The keyword **pm** stands for pattern matching, and M **to** $x.N$ denotes the sequencing of computations: if M returns a value, then N is executed, with the value bound to x. We refer to [5] for a motivation of the choice of these constructs. (The notation in *loc. cit.* differs from ours in that application $M\,V$ is written in reverse order as V'M.)

We write $\mathrm{Val}(A)$ for the set of (closed) values V such that $\emptyset \vdash^{\mathsf{v}} V : A$. Likewise, we define $\mathrm{Comp}(\underline{B})$ as the set of computations M where $\emptyset \vdash^{\mathsf{c}} M : \underline{B}$.

2.1.2 Stacks and stack typing

Stacks were introduced to CBPV as a conceptual device for structuring the semantics, and also appear naturally

$$\frac{}{\Gamma \vdash^{s} \mathbf{nil} : \underline{C} \multimap \underline{C}}$$

$$\frac{\Gamma, x{:}A \vdash^{c} M : \underline{B} \qquad \Gamma \vdash^{s} K : \underline{B} \multimap \underline{C}}{\Gamma \vdash^{s} [\cdot]\ \mathbf{to}\ x.M :: K : FA \multimap \underline{C}}$$

$$\frac{\Gamma \vdash^{s} K : \underline{B}_j \multimap \underline{C}}{\Gamma \vdash^{s} j :: K : \prod_{i \in I} \underline{B}_i \multimap \underline{C}} (j \in I)$$

$$\frac{\Gamma \vdash^{v} V : A \qquad \Gamma \vdash^{s} K : \underline{B} \multimap \underline{C}}{\Gamma \vdash^{s} V :: K : (A \to \underline{B}) \multimap \underline{C}}$$

Fig. 2. Stacks and stack typing

in abstract machine-based operational semantics [5,6]. Intuitively, stacks represent the context of a CBPV computation, and thus there are stack frames corresponding to the sequential composition of computations, and projections from products, and to function arguments. They are defined formally in Figure 2, where $\underline{B} \multimap \underline{C}$ denotes the type of stacks that represent computations of type $\underline{C}$ once a computation of type $\underline{B}$ is plugged in. We write $\mathrm{Stk}(\underline{B})$ for the set of stacks K such that $\emptyset \vdash^{s} K : \underline{B} \multimap \underline{C}$ for some $\underline{C}$.

The *dismantling* of a stack is defined by moving frames from the stack onto the term:

$$\begin{aligned} M \bullet \mathbf{nil} &= M \\ M \bullet ([\cdot]\ \mathbf{to}\ x.N :: K) &= (M\ \mathbf{to}\ x.N) \bullet K \\ M \bullet j :: K &= (M\ j) \bullet K \\ M \bullet V :: K &= (M\ V) \bullet K \end{aligned}$$

This operation is employed as a tool for reasoning about stacks in [6], along with the concatenation of stacks. We will also make heavy use of dismantling below when lifting reducibility predicates from value types to computation types. It is easy to see that $\Gamma \vdash^{c} M : \underline{B}$ and $\Gamma \vdash^{s} K : \underline{B} \multimap \underline{C}$ implies $\Gamma \vdash^{c} M \bullet K : \underline{C}$.

2.2 Operational Semantics

Several concrete operational semantics of CBPV have been considered in previous work, amongst them a big-step semantics, and an (equivalent) formulation in terms of a CK machine. The reduction rules we consider here are derived from the equational theory of CBPV as presented in [5], which contains both these semantics, and is itself preserved by all the denotational models of CBPV. The rules are collected in Figure 3, and fall into three categories described in more detail next.

β reductions

$$\begin{aligned} \mathbf{force}\ (\mathbf{thunk}\ M) &\longrightarrow M \\ \mathbf{pm}\ \langle j, V \rangle\ \mathbf{as}\ \{\ldots, \langle i, x \rangle .M_i, \ldots\} &\longrightarrow M_j[V\,/\,x] \\ \mathbf{pm}\ (V, W)\ \mathbf{as}\ (x, y).M &\longrightarrow M[V\,/\,x, W\,/\,y] \\ \mathbf{return}\ V\ \mathbf{to}\ x.M &\longrightarrow M[V\,/\,x] \\ \lambda\{\ldots, i.M_i, \ldots\}\ j &\longrightarrow M_j \\ (\lambda x.M)\ V &\longrightarrow M[V\,/\,x] \end{aligned}$$

η reductions (where $x, y \notin fv(M)$)

$$\begin{aligned} \mathbf{thunk}\ (\mathbf{force}\ V) &\longrightarrow V \\ \mathbf{pm}\, V\, \mathbf{as}\ \{\ldots, \langle i, x \rangle .M[\langle i, x \rangle\,/\,z], \ldots\} &\longrightarrow M[V\,/\,z] \\ \mathbf{pm}\ V\ \mathbf{as}\ (x, y).M[(x, y)\,/\,z] &\longrightarrow M[V\,/\,z] \\ M\ \mathbf{to}\ x.(\mathbf{return}\ x) &\longrightarrow M \\ \lambda\{\ldots, i.(M\ i), \ldots\}\ j &\longrightarrow M \\ \lambda x.(M\ x) &\longrightarrow M \end{aligned}$$

permutative reductions (where $x \notin fv(M'', V)$)

$$\begin{aligned} (M\ \mathbf{to}\ x.M')\ \mathbf{to}\ y.M'' &\longrightarrow M\ \mathbf{to}\ x.(M'\ \mathbf{to}\ y.M'') \\ (M\ \mathbf{to}\ x.M')\ j &\longrightarrow M\ \mathbf{to}\ x.(M\ j) \\ (M\ \mathbf{to}\ x.M')\ V &\longrightarrow M\ \mathbf{to}\ x.(M\ V) \end{aligned}$$

Fig. 3. Operational semantics

2.2.1 β reductions

There are six β rules, one for each type constructor (there is none for **1**). As mentioned above, values of sum and product types are deconstructed using pattern matching. Projection pairs, on the other hand, are similar to procedures except that they are applied to a tag and correspondingly no binding takes place.

2.2.2 η reductions

There are also six η rules (again there is none for terms of type **1**). Note that the rules for pattern matching assume that all occurrences of the pattern variables in M are again in the form of the pattern.

2.2.3 Permutative reductions

Permutative conversions appear in many calculi with binding constructs other than lambdas, and have 'administrative' character: they serve to rearrange parenthesis, thus enabling further reductions. Their logical significance, via the Curry-Howard correspondence, is to establish the subformula property with respect to natural deduction proofs e.g. in the presence of disjunctions.

In the equational theory of CBPV there are three such permutative conversions which distribute the elimina-

tion construct $[\cdot]$ **to** $x.M'$ for returner types FA over those of the other computation types.

2.2.4 Reductions on stacks

The reduction rules on terms induce a reduction on stacks, by

$$K \longrightarrow K' \iff \forall M.\, M \bullet K \longrightarrow M \bullet K'$$

Intuitively, $K \longrightarrow K'$ if this reduction is possible independent of the particular term M plugged in. The length $|K|$ of a stack $K = F_n :: \ldots :: F_1 :: \mathbf{nil}$ is n.

Lemma 1 (Stack length) *For all K, K', if $K \longrightarrow K'$ then $|K| \geq |K'|$.*

PROOF. Since the reduction cannot depend on a specific term plugged into K, it must be caused either by reduction within a single stack frame, or by an interaction between adjacent stack frames. In the former case, it is clear that the length cannot increase (it may decrease, though, in the case where $K = F_n :: \ldots :: [\cdot]$ **to** $x.\mathbf{return}\ x :: K'' \longrightarrow F_n :: \ldots :: K''$ by an η reduction). The only interaction between stack forms is due to the permutative reductions, which also give $|K'| = |K| - 1$. ∎

3 Strong Normalization

We write $\mathcal{SN}$ for the set of strongly normalizing terms (either values, computations, or stacks). Since $\longrightarrow$ is finitely branching, we can associate to each term $P \in \mathcal{SN}$ the length $\nu(P) \in \mathbb{N}$ of a longest reduction sequence from P. Note that $P[Q/x] \in \mathcal{SN}$ implies that $P \in \mathcal{SN}$.

3.1 Reducibility for Computation Types and Stacks

For every computation type $\underline{B}$ we consider the pair of operations

$$\begin{aligned}
&(\cdot)^\top : \mathcal{P}(\mathrm{Comp}(\underline{B})) \to \mathcal{P}(\mathrm{Stk}(\underline{B}))\\
&\quad T^\top = \{K \in \mathrm{Stk}(\underline{B}) \mid \forall M \in T.\, M \bullet K \in \mathcal{SN}\}\\
&(\cdot)^\top : \mathcal{P}(\mathrm{Stk}(\underline{B})) \to \mathcal{P}(\mathrm{Comp}(\underline{B}))\\
&\quad S^\top = \{M \in \mathrm{Comp}(\underline{B}) \mid \forall K \in S.\, M \bullet K \in \mathcal{SN}\}
\end{aligned}$$

It is easy to see that these operations form a Galois connection between the complete lattices of sets of computations and sets of stacks, ordered by inclusion: for all T and S,

$$T \subseteq S^\top \iff T^\top \supseteq S$$

Thus they induce a closure operation $T \mapsto T^{\top\top}$, and we will make use of the following property below.

$$\begin{aligned}
\mathrm{red}^{\mathsf{v}}_{U\underline{B}} &= \{\mathbf{thunk}\ M \,|\, M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}\}\\
\mathrm{red}^{\mathsf{v}}_{\sum_{i\in I} A_i} &= \{\langle j, V\rangle \,|\, j \in I, V \in \mathrm{red}^{\mathsf{v}}_{A_j}\}\\
\mathrm{red}^{\mathsf{v}}_{\mathbf{1}} &= \{\star\}\\
\mathrm{red}^{\mathsf{v}}_{A\times A'} &= \{(V, V') \mid V \in \mathrm{red}^{\mathsf{v}}_{A}, V' \in \mathrm{red}^{\mathsf{v}}_{A'}\}\\
\mathrm{red}^{\mathsf{c}}_{\underline{B}} &= (\mathrm{red}^{\mathsf{s}}_{\underline{B}})^\top\\
\mathrm{red}^{\mathsf{s}}_{FA} &= \{\mathbf{return}\ V \,|\, V \in \mathrm{red}^{\mathsf{v}}_{A}\}^\top\\
\mathrm{red}^{\mathsf{s}}_{\prod_{i\in I}\underline{B}_i} &= \{\lambda\{\ldots, i.M_i, \ldots\} \,|\, \forall j.\, M_j \in \mathrm{red}^{\mathsf{c}}_{\underline{B}_j}\}^\top\\
\mathrm{red}^{\mathsf{s}}_{A\to\underline{B}} &= \{\lambda x.M \,|\, \forall V \in \mathrm{red}^{\mathsf{v}}_{A}.\, M[V/x] \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}\}^\top
\end{aligned}$$

Fig. 4. Reducibility candidates

Fact 2 (Extensiveness) *The operation $(\cdot)^{\top\top}$ is extensive, i.e., $T \subseteq T^{\top\top}$ for every $T \subseteq \mathit{Comp}(\underline{B})$.* ∎

Figure 4 defines type-indexed families $\mathrm{red}^{\mathsf{v}}_A$ and $\mathrm{red}^{\mathsf{c}}_{\underline{B}}$ of *reducible terms*, by induction on types. In fact, this is achieved by simultaneously defining a similar family of reducibility candidates $\mathrm{red}^{\mathsf{s}}_{\underline{B}}$ on stacks. In the following we can restrict attention to closed terms, due to the following observation (which relies on the extensiveness of the operation $T \mapsto T^{\top\top}$).

Fact 3 (Inhabitation) *For all types A and $\underline{B}$ there are $V \in \mathit{Val}(A)$ and $M \in \mathit{Comp}(\underline{B})$ such that $V \in \mathrm{red}^{\mathsf{v}}_A$ and $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$, resp.* ∎

3.2 Proving Strong Normalization

We proceed 'as usual', by first proving that reducibility implies strong normalization, is reduction-closed, and expansion-closed. A computation M is *neutral* if it is not an introduction form, i.e., if it is not one of **return** V, $\lambda\{\ldots, i.M_i, \ldots\}$, or $\lambda x.M'$.

Lemma 4 *For all value types A the following holds:*

(V1) If $V \in \mathrm{red}^{\mathsf{v}}_A$ then $V \in \mathcal{SN}$.
(V2) If $V \in \mathrm{red}^{\mathsf{v}}_A$ and $V \longrightarrow V'$ then $V' \in \mathrm{red}^{\mathsf{v}}_A$.

Similarly, for all computation types $\underline{B}$:

(S1) If $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$ then $K \in \mathcal{SN}$.
(S2) $\mathbf{nil} \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$.

and

(C1) If $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ then $M \in \mathcal{SN}$.
(C2) If $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ and $M \longrightarrow M'$ then $M' \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$.
(C3) If $M' \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ for all M' such that $M \longrightarrow M'$, and M is neutral, then $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$.

PROOF. The properties are proved simultaneously by induction on the type. We establish (V1) and (V2) first.

- For $\mathbf{1}$ this is immediate since $\star$ is the only term to consider.
- For $U\underline{B}$ assume that $V = \mathbf{thunk}\ M$. If $V \in \mathrm{red}^{\mathsf{v}}_{U\underline{B}}$ then $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$, thus $M \in \mathcal{SN}$ by part (C1) of the induction hypothesis. We show $V \in \mathcal{SN}$ by induction on $\nu(M)$. If M is not of the form $\mathbf{force}\ W$ then $V \longrightarrow V'$ only if $V' = \mathbf{thunk}\ M'$ and $M \longrightarrow M'$. But then $\nu(M') < \nu(M)$ and the result follows by induction hypothesis. If $M = \mathbf{force}\ W$ then there is additionally the η reduction $V \longrightarrow W$ possible. Now $W \in \mathrm{Val}(U\underline{B})$ implies that $W = \mathbf{thunk}\ N$ for some N. But then also $M = \mathbf{force}(\mathbf{thunk}\ N) \longrightarrow N$ by β reduction. In particular, $\nu(N) < \nu(M)$ and we can apply the induction hypothesis again.

  A similar case distinction on the reducts of V shows that (V2) holds, using part (C2) of the induction hypothesis for the subterm M.
- For $\sum_{i\in I} A_i$ and $A \times A'$, (V1) and (V2) follow by induction hypothesis, using the fact that all reductions are necessarily found in proper subterms of $V \in \mathrm{red}^{\mathsf{v}}_{\sum_{i\in I} A_i}$ and $V \in \mathrm{red}^{\mathsf{v}}_{A\times A'}$, respectively.

Next, for computation types, we consider (S1). Assume $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$, then by Fact 3 there exists some $N \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$. Thus, $N \bullet K \in \mathcal{SN}$ by definition of $\mathrm{red}^{\mathsf{c}}_{\underline{B}} = (\mathrm{red}^{\mathsf{s}}_{\underline{B}})^{\top}$. Since $K \longrightarrow K'$ only if for all N, $N \bullet K \longrightarrow N \bullet K'$, the strong normalization of K follows.

We prove (S2):

- Since for all $V \in \mathrm{Val}(A)$, $V \in \mathrm{red}^{\mathsf{v}}_{A}$ implies $V \in \mathcal{SN}$ by part (V1) of the induction hypothesis, we obtain $\mathbf{return}\ V \bullet \mathbf{nil} = \mathbf{return}\ V \in \mathcal{SN}$. Hence, $\mathbf{nil} \in \mathrm{red}^{\mathsf{s}}_{FA}$.
- To show $\mathbf{nil} \in \mathrm{red}^{\mathsf{s}}_{A\to\underline{B}}$, let $M = \lambda x.N \in \mathrm{Comp}(A \to \underline{B})$ be such that $N[V/x] \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ for all $V \in \mathrm{red}^{\mathsf{v}}_{A}$. By part (C1) of the induction hypothesis, $N[V/x] \in \mathcal{SN}$. From this it is easy to conclude that $M \bullet \mathbf{nil} = M \in \mathcal{SN}$, which was to show.
- Assume $M = \lambda\{\ldots, i.M_i, \ldots\} \in \mathrm{Comp}(\prod_{i\in I} \underline{B}_i)$ is such that $M_i \in \mathrm{red}^{\mathsf{c}}_{\underline{B}_i}$ for all i. Thus by induction $M_i \in \mathcal{SN}$ which implies $M \bullet \mathbf{nil} = M \in \mathcal{SN}$.

Now (C1)–(C3) can be proved uniformly for all computation types: property (C1) follows directly since $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ and (S2) imply, by definition of $\mathrm{red}^{\mathsf{c}}_{\underline{B}} = (\mathrm{red}^{\mathsf{s}}_{\underline{B}})^{\top}$, that $M = M \bullet \mathbf{nil} \in \mathcal{SN}$.

Next, assume $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ and $M \longrightarrow M'$. Let $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$; to establish (C2) we must show that $M' \bullet K \in \mathcal{SN}$. But since $M \longrightarrow M'$ implies $M \bullet K \longrightarrow M' \bullet K$ this is just a consequence of $M \bullet K \in \mathcal{SN}$ which follows from the reducibility of M.

Finally, to prove (C3), assume that M is neutral and $M' \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ for all M' such that $M \longrightarrow M'$, and let $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$. As seen in the proof of (S1) above, K is strongly normalizing so that we can use induction on $\nu(K)$ to prove $M \bullet K \in \mathcal{SN}$, from which $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ follows since $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$ was chosen arbitrarily. Since M is assumed neutral, the only possible reduction descendants of $M \bullet K$ are $M' \bullet K$ and $M \bullet K'$. The former is strongly normalizing as $M' \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ and $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$, while for the latter we have $\nu(K') < \nu(K)$ and thus the induction hypothesis applies. ∎

Next, we aim to show that every typable, closed term is reducible. To this end, we must generalize to terms with free variables in order to obtain an induction hypothesis that is sufficiently strong. This gives the following statement, corresponding to the 'basic lemma' of logical relations for lambda calculus [8].

Lemma 5 (Fundamental property) *Suppose that* $\Gamma = x_1{:}A_1, \ldots x_n{:}A_n$ *and* $V_i \in \mathrm{red}^{\mathsf{v}}_{A_i}$ *for all* $i = 1, \ldots, n$.

(1) If $\Gamma \vdash^{v} V : A$ *then* $V[\vec{V}/\vec{x}] \in \mathrm{red}^{\mathsf{v}}_{A}$.
(2) If $\Gamma \vdash^{c} M : \underline{B}$ *then* $M[\vec{V}/\vec{x}] \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$.

By parts (V1) and (C1) of Lemma 4 it is therefore immediate that closed typable terms are strongly normalizing. This extends to open terms by observing that $V \in \mathcal{SN}$ iff $\mathbf{return}\ V \in \mathcal{SN}$, and $M \in \mathcal{SN}$ iff $\lambda\vec{x}.M \in \mathcal{SN}$. Thus we have shown:

Theorem 6 (Strong normalization) *Reduction in CBPV is strongly normalizing.* ∎

It remains to prove Lemma 5, which we do in the following.

Lemma 7 *If* $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}\multimap\underline{C}}$ *and* $M[V/x] \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ *for all* $V \in \mathrm{red}^{\mathsf{v}}_{A}$, *then* $[\cdot]\ \mathbf{to}\ x.M :: K \in \mathrm{red}^{\mathsf{s}}_{FA\multimap\underline{C}}$.

PROOF. Let $V \in \mathrm{red}^{\mathsf{v}}_{A}$. We must show that $N = \mathbf{return}\ V \bullet ([\cdot]\ \mathbf{to}\ x.M :: K) = (\mathbf{return}\ V\ \mathbf{to}\ x.M) \bullet K \in \mathcal{SN}$. By assumption and Lemma 4 (V1,C1), both $V \in \mathcal{SN}$ and $M[V/x] \bullet K \in \mathcal{SN}$, and we proceed by induction on $\nu(V) + \nu(M \bullet K) + |K|$. Consider the reductions of N:

- By a β reduction, $N \longrightarrow M[V/x] \bullet K$ which is strongly normalizing.
- If $M = \mathbf{return}\ x$ then, by an η reduction, $N \longrightarrow \mathbf{return}\ V \bullet K$. This is just $M[V/x] \bullet K$ thus strongly normalizing.
- If $K = [\cdot]\ \mathbf{to}\ y.P :: K'$ then, by a permutative reduction, $N \longrightarrow \mathbf{return}\ V \bullet ([\cdot]\ \mathbf{to}\ x.(M\ \mathbf{to}\ y.P) :: K')$. Thus $|K'| < |K|$, and since $M\ \mathbf{to}\ y.P \bullet K' = M \bullet K$ the induction hypothesis applies.
- If $K = j :: K'$ then, by a permutative reduction, $N \longrightarrow \mathbf{return}\ V \bullet ([\cdot]\ \mathbf{to}\ x.M\ j :: K')$. Again $|K'| < |K|$ and $M\ j \bullet K' = M \bullet K$ means the induction hypothesis is applicable.

- The case where $K = W :: K$ and N reduces by a permutative conversion to **return** $V \bullet ([\cdot]$ **to** $x.M\ W :: K')$ is similar.
- All other reductions of N are confined to either V, M or K, and thus strictly decrease either $\nu(V)$ or $\nu(M \bullet K)$. Moreover, by Lemma 1, $|K|$ does not increase, so the induction hypothesis applies. ∎

Lemma 8 *If* $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}}$ *and* $V \in \mathrm{red}^{\mathsf{v}}_{A}$ *then* $V :: K \in \mathrm{red}^{\mathsf{s}}_{A \to \underline{B}}$.

PROOF. Suppose $M = \lambda x.N$ is such that $N[V/x] \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ for all $V \in \mathrm{red}^{\mathsf{v}}_{A}$. We must show that $M \bullet (V :: K) \in \mathcal{SN}$. By Definition of $\mathrm{red}^{\mathsf{c}}_{A \to \underline{B}}$, the extensiveness of $(\cdot)^{\top\top}$ and Lemma 4 (C1), $M \in \mathcal{SN}$ and we can proceed by induction on $\nu(M) + \nu(V) + \nu(K)$. The only case that is not directly handled by the induction hypothesis is the β reduction $M \bullet (V :: K) \longrightarrow N[V/x] \bullet K$. But now reducibility of K and $N[V/x]$ imply strong normalization of $N[V/x] \bullet K$, by definition of $\mathrm{red}^{\mathsf{s}}_{\underline{B}}$. ∎

Similarly, it is not difficult to prove

Lemma 9 *Suppose* $K \in \mathrm{red}^{\mathsf{s}}_{\underline{B}_k}$ *and* $j \in I$. *Then* $j :: K \in \mathrm{red}^{\mathsf{s}}_{\prod_{i \in I} \underline{B}_i}$. ∎

PROOF of Lemma 5. The properties are proved simultaneously, by induction on the typing derivations. We distinguish cases, depending on the last typing rule applied in the derivation.

- If $\Gamma \vdash^{\mathsf{v}} x_i : A_i$ then clearly $V_i \in \mathrm{red}^{\mathsf{v}}_{A_i}$ by assumption.
- The case $\Gamma \vdash^{\mathsf{v}} \star : \mathbf{1}$ is immediate by definition of $\mathrm{red}^{\mathsf{v}}_{\mathbf{1}}$.
- The cases $\Gamma \vdash^{\mathsf{v}} \langle j, V \rangle : \sum_{i \in I} A_i$, $\Gamma \vdash^{\mathsf{v}} (V, V') : A \times A'$ and $\Gamma \vdash^{\mathsf{v}}$ **thunk** $M : U\underline{B}$ follow by induction hypothesis and the definition of the respective predicate.

- The cases $\Gamma \vdash^{\mathsf{c}}$ **return** $V : FA$, $\Gamma \vdash^{\mathsf{c}} \lambda x.M : A \to \underline{B}$ and $\Gamma \vdash^{\mathsf{c}} \lambda\{\ldots, i.M_i, \ldots\} : \prod_{i \in I} \underline{B}_i$ follow by induction hypothesis, extensiveness of $(\cdot)^{\top\top}$ and the definition of the respective predicate.
- If $\Gamma \vdash^{\mathsf{c}}$ **force** $V : \underline{B}$ then $W = V[\vec{V}/\vec{x}] \in \mathrm{red}^{\mathsf{v}}_{U\underline{B}}$ by induction hypothesis, so that by Lemma 4 (V1), $W \in \mathcal{SN}$. We prove **force** $W \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ by induction on $\nu(W)$. By definition of $\mathrm{red}^{\mathsf{v}}_{U\underline{B}}$, $W =$ **thunk** M for some $M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$, and if **force** $W \longrightarrow N$ then either $N = M \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$, or else $N =$ **force** W' for some W'. By Lemma 4 (V2), $W' \in \mathrm{red}^{\mathsf{v}}_{U\underline{B}}$, and since $\nu(W') < \nu(W)$ we have $N \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$ also in this case. By Lemma 4 (C3), **force** $W \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$.
- If $\Gamma \vdash^{\mathsf{c}}$ **pm** V **as** $(x, y).M : \underline{B}$ then by induction hypothesis and definition of $\mathrm{red}^{\mathsf{v}}_{A \times A'}$, $V[\vec{V}/\vec{x}] = (W, W')$ for some $W \in \mathrm{red}^{\mathsf{v}}_{A}$ and $W' \in \mathrm{red}^{\mathsf{v}}_{A'}$, thus $M[\vec{V}, W, W'/\vec{x}, x, y] \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$. By Lemma 4 (V1,C1) we can use induction on $\nu(W) + \nu(W') + \nu(M[\vec{V}/\vec{x}])$ to prove that $N =$ **pm** (W, W') **as** $(x, y).M[\vec{V}/\vec{x}]$ is reducible. Consider its possible reducts:
  - If $N \longrightarrow M[\vec{V}, W, W'/\vec{x}, x, y]$ by β then this has been established above.
  - If $M[\vec{V}/\vec{x}]$ is of the form $P[(x, y)/z]$ for some P such that $x, y \notin fv(P)$ then $N \longrightarrow P[(W, W')/z] = M[\vec{V}, W, W'/\vec{x}, x, y]$ by η reduction, and again this is reducible.
  - If $N \longrightarrow N'$ by some reduction within W, W' or $M[\vec{V}/\vec{x}]$ then reducibility follows by induction hypothesis.

  There are no other reducts, hence by Lemma 4 (C3) also $N \in \mathrm{red}^{\mathsf{c}}_{\underline{B}}$.
- The case $\Gamma \vdash^{\mathsf{c}}$ **pm** V **as** $\{\ldots, \langle j, x \rangle .M_j, \ldots\} : \underline{B}$ is proven similarly, showing by induction on $\nu(V[\vec{V}/\vec{x}]) + \sum_{i \in I} \nu(M_i[\vec{V}/\vec{x}])$ that all reductions lead to reducible terms and then applying Lemma 4 (C3).
- If $\Gamma \vdash^{\mathsf{c}} M$ **to** $x.N : \underline{B}$ then we must show, for any $K \in \mathrm{red}^{\mathsf{s}}_{FA}$, that M **to** $x.N \bullet K \in \mathcal{SN}$ or equivalently, that $M \bullet ([\cdot]$ **to** $x.N :: K) \in \mathcal{SN}$. But this is clear, since $M \in \mathrm{red}^{\mathsf{c}}_{FA}$ and $[\cdot]$ **to** $x.N :: K \in \mathrm{red}^{\mathsf{s}}_{FA}$ by induction hypothesis and Lemma 7 above.
- The cases $\Gamma \vdash^{\mathsf{c}} M\ V : \underline{B}$ and $\Gamma \vdash M\ j : \underline{B}_j$ are similarly handled by induction and Lemma 8 and Lemma 9, after pushing V and j, resp., onto the stack. ∎

References

[1] N. Benton, G. M. Bierman, V. de Paiva, Computational types from a logical perspective, Journal of Functional Programming 8 (2) (1998) 177–193.

[2] P. de Groote, On the strong normalisation of intuitionistic natural deduction with permutation-conversions., Information and Computation 178 (2) (2002) 441–464.

[3] J.-Y. Girard, Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur, Thèse de doctorat d'état, Université de Paris VII (1972).

[4] P. B. Levy, Call-by-push-value: A subsuming paradigm, in: J.-Y. Girard (ed.), Proceedings Typed Lambda Calculi and Applications (TLCA'99), vol. 1581 of Lecture Notes in Computer Science, Springer, 1999.

[5] P. B. Levy, Call-By-Push-Value, vol. 2 of Semantic Structures in Computation, Kluwer, 2004.

[6] P. B. Levy, Adjunction models for call-by-push-value with stacks, Theory and Applications of Categories 14 (2005) 75–110.

[7] S. Lindley, I. Stark, Reducibility and $\top\top$-lifting for computation types, in: P. Urzyczyn (ed.), Typed Lambda Calculi and Applications (TLCA'05), vol. 3461 of Lecture Notes in Computer Science, Springer, 2005.

[8] J. C. Mitchell, Foundations for Programming Languages, MIT Press, 1996.

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

[10] D. Prawitz, Ideas and results in proof theory, in: J. E. Fenstad (ed.), Proceedings of the Second Scandinavian Logic Symposium, vol. 63 of Studies in Logic and the Foundations of Mathematics, North-Holland, 1971.

[11] W. W. Tait, Intensional interpretation of functionals of finite type I, Journal of Symbolic Logic 32 (2) (1967) 198–212.