Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Bachelor's Program in Computer Science

Bachelor's Thesis

Strong Normalization of Call-By-Push-Value

submitted by

Christian Doczkal

on September 13, 2007

Supervisor

Prof. Dr. Gert Smolka

Advisor

Dr. Jan Schwinghammer

Reviewers

Prof. Dr. Gert Smolka

Prof. Dr. Holger Hermanns

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, September 13, 2007

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, September 13, 2007

Abstract

Strong normalization is one of the fundamental proof-theoretic aspects of many typed λ-calculi. We prove strong normalization of Call-By-Push-Value with respect to a small-step reduction relation based on the equational theory of the calculus. To obtain a uniform treatment of the various computation types of Call-By-Push-Value, we adapt Lindley's $\top\top$-lifting to a notion of $\top\top$-closure. This allows us to define a Girard/Tait style reducibility predicate, in the presence of several different computation types.

Acknowledgements

I am very much in debt to my advisor, Jan Schwinghammer, for his continuous support during the last months. I thank Jan for his patience when explaining basic concepts to me and for helping me understand some of the subtleties we encountered. I am especially grateful to Jan for always being there if I needed help or advice.

I thank Gert Smolka for his introduction to logics, which inspired me to select this topic. I also thank Gert Smolka and Holger Hermanns for reviewing this thesis.

Finally, I want to thank all my friends for their support, especially on those occasions where things did not work out as planned.

Contents

Chapter 1

Introduction

The Call-By-Push-Value calculus is a model of higher-order computation and a refinement of the simply-typed λ-calculus that can easily be adapted to various computational effects. In this thesis, we will provide a notion of reduction for the Call-By-Push-Value calculus, based on its equational therory, and prove strong normalization of the resulting calculus.

When studying the pure simply-typed λ-calculus (see Table 1.1), evaluation order does not matter. Every term reduces to a unique normal form, regardless of the order in which the reduction rules are applied. This no longer holds, once computational effects, like divergence or output, are added. Consider the term $(\lambda x.1)$ `diverge`, where `diverge` is a term that has no normal form. Under call-by-value (CBV), where functions are only applied to fully evaluated arguments, this term has no normal form. Under call-by-name (CBN), where functions are applied only to completely unevaluated terms, the term reduces to 1. If we add printing, every term still has a normal form, but the output of a term like $(\lambda x.\texttt{print}\text{“a”}; x)$ $(\texttt{print}\text{“b”}; 1)$ is “ab” under CBN but “ba” under CBV.

Thus, whenever a new style of semantics is defined it usually has to be defined individually for the call-by-name and the call-by-value setting, as the two are fundamentally different. Furthermore, many results need to be proven for both settings individually.

1.1 Call-By-Push-Value

This duplication of work led to the introduction of the Call-By-Push-Value (CBPV) calculus by Paul B. Levy [Lev99]. The main idea behind the CBPV calculus is to serve as a unified paradigm, which is meant to “subsume” the simply-typed λ-calculus under CBV and CBN in the presence of various computational effects. Subsumption here means that there are embeddings from the simply typed λ-calculus, one for CBN and one for CBV, into CBPV, such that all the usual operational semantics, denotational semantics, or

Terms and typing rules

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}$$

β-reductions

$$(\lambda x.M)\ N \to M[N/x] \qquad \pi_1(M, N) \to M \qquad \pi_2(M, N) \to N$$

η-reductions

$$\lambda x.M\ x \to M \qquad (\pi_1 M, \pi_2 M) \to M$$

Table 1.1: The simply-typed λ-calculus

equations for CBN and CBV can be derived, via the embeddings, from the corresponding properties of CBPV. To accomplish this, the calculus needs to be very fine grain. It distinguishes values from computations already at type level. A term of type $F\,A$ for example is a computation that will return a value of type A, possibly causing effects in the process. CBPV functions are also computations, but their arguments are always values. So to apply one function to another, we have to build a *thunk* of the argument, thus "wrapping" the computation into a value. So an application $f\ g$ in the λ-calculus corresponds to $f(\texttt{thunk}\ g)$ in CBPV. Extensive work on the calculus and its relations to other calculi can be found in [Lev04].

In this thesis, we will investigate the CBPV calculus in an effect free setting. To subsume the simply-typed λ-calculus under the deterministic reduction relations CBN and CBV, Levy also defines the Call-By-Push-Value calculus using a deterministic reduction relation. We will generalize the notion of reduction in CBPV to allow reductions to occur anywhere in a term, basing our reduction relation on the CBPV equational theory. This opens up classic proof theoretic questions like strong normalization and confluence of the new reduction relation.

1.2 Strong Normalization

Proving strong normalization turns out to be interesting and non-trivial due to the computation types of the calculus and the elimination constructs associated with them. These make a direct application of the Girard/Tait

proof technique (cf. Section 3.1) rather difficult. There are normalization results for other computational calculi, like Moggi's computational metalanguage λ_{ml} [Mog91]. The calculus λ_{ml}, however, has only one additional type constructor for computation types. CBPV on the other hand has several different kinds of computation types, like the ones noted above, so a uniform treatment of those is desirable.

To obtain a treatment of all computation types that is as uniform as possible, we will introduce the notion of a $\top\top$-closure. This will be a variant of the $\top\top$-lifting approach of Lindley and Stark used to prove strong normalization of λ_{ml} [LS05]. We will then show how this adaptation can be used to prove strong normalization of CBPV.

Furthermore, we will investigate the embeddings from CBN and CBV into CBPV in order to derive a relation between the operational semantics of the λ-calculus under free $\beta\eta$-reduction and our reduction relation for CBPV. We will show how to derive the well known normalization result for the lambda calculus from our normalization result for CBPV.

At last, we will investigate confluence of CBPV. As it turns out, there are two reduction rules in CBPV that each cause confluence to fail. We will provide an example and relate this to a similar problem arising in the simply-typed λ-calculus with sums.

1.3 Outline

In Chapter 2 we introduce the Call-By-Push-Value calculus as it was originally introduced by P.B. Levy along with two equivalent operational semantics, as well as CBPV stacks. Furthermore, we develop a nondeterministic reduction relation based on the CBPV equational theory.

In Chapter 3 we give an overview over the field of strong normalization. We introduce the classic Girard/Tait technique, used to prove strong normalization of the simply-typed λ-calculus, which is also the basis for our normalization proof. We explain the problem of a direct application of the Girard/Tait technique to computational calculi and consider the approach of $\top\top$-lifting, which can be used to handle computational or monadic types in normalization proofs.

In Chapter 4 we adapt the approach of $\top\top$-lifting to define our notion $\top\top$-closure. We then use this to prove strong normalization of CBPV.

In Chapter 5 we investigate possible embeddings of the λ-calculus under unrestricted $\beta\eta$-reduction into CBPV with our nondeterministic reduction relation. We also show why confluence fails for CBPV and introduce directions for further work.

Chapter 2

The Call-By-Push-Value Calculus

The Call-By-Push-Value calculus was originally introduced by Paul Blain Levy in 1999 [Lev99]. The idea that led to the introduction of CBPV was that when investigating the simply-typed λ calculus under fixed evaluation strategies, like call-by-name and call-by-value, many results need to be proven for each such evaluation strategy individually. CBPV was introduced to "subsume" the simply-typed λ-calculus under call-by-name and call-by-value.

2.1 Terms and Types

To achieve this, the calculus needs to be very fine grain. The central idea is to distinguish values from computations, both at term and at type level. The main intuition behind this is that values "are", and thus never need to be evaluated, while computations "do" [Lev99]. Thus, CBPV has two type judgements $\Gamma \vdash^v V : A$ assigning value types to value terms and $\Gamma \vdash^c M : \underline{B}$ assigning computation types to computations. When denoting types in CBPV, we use the following conventions:

Notation. We always denote computation types with an underscore. Thus, $A, A', \ldots$ stand for value types, while $\underline{B}, \underline{C}, \ldots$ are computation types. For terms U, V, W range over values while M, N range over computations.

Using this notation, we have the following types in CBPV:

$$A ::= U\,\underline{B} \mid \Sigma_{i\in I} A_i \mid A \times A \mid 1$$
$$\underline{B} ::= F\,A \mid \Pi_{i\in I} \underline{B}_i \mid A \to \underline{B}$$

where I is any countable set of indices.

For values, there is the thunk type $U\,\underline{B}$, representing a computation wrapped into a value. Then we have sum types with countable sets of

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

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

$$\frac{\Gamma \vdash^c N : \underline{B}}{\Gamma \vdash^v \texttt{thunk}\ N : U\,\underline{B}} \qquad \frac{\Gamma \vdash^v V : U\,\underline{B}}{\Gamma \vdash^c \texttt{force}\ V : \underline{B}}$$

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

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

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

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

Table 2.1: Typing rules for the terms from CBPV

indices, as well as pairs of values. 1 is the unit type, whose only value is $()$. For computations, there is the returner type $F\,A$ for computations returning a value of type A and a projection product, which is an indexed collection of computations. Furthermore, we have the usual function types with the restriction that in CBPV all functions go from value types to computation types.

The types mentioned so far allow some interesting basic types to be defined within the CBPV calculus. For example, the type `bool` can be defined as $1 + 1 = \sum_{i \in \{0,1\}} 1$, and `nat` can be defined as $\sum_{i \in \mathbb{N}} 1$.

The terms of the calculus and the corresponding typing rules can be found in Table 2.1. Here Γ is a context defined as follows.

Definition 2.1.1 (Context)**.** A context Γ is a finite list of identifiers with associated value types $x_0 : A_0, \ldots, x_k : A_k$.

Note that for both judgements, only value types appear to the left of $\vdash$. Thus, the restriction on function types is necessary, since function application is done by substitution and variables can only have value type. In the elimination rules for sums and pairs, `pm` stands for "pattern matching" of the corresponding values into a computation. The semantics will be shown in the next section.

$$\frac{}{\texttt{return}\ V \Downarrow \texttt{return}\ V} \qquad \frac{}{\lambda\{\ldots, i.M_i, \ldots\} \Downarrow \lambda\{\ldots, i.M_i, \ldots\}}$$

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

$$\frac{M \Downarrow T}{\texttt{force}\ \texttt{thunk}\ M \Downarrow T} \qquad \frac{M \Downarrow \texttt{return}\ V \qquad N[V/x] \Downarrow T}{M\ \texttt{to}\ x.N \Downarrow T}$$

$$\frac{M[V/x, V'/y] \Downarrow T}{\texttt{pm}\ (V, V')\ \texttt{as}\ (x, y).M \Downarrow T} \qquad \frac{M_i[V/x] \Downarrow T}{\texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \{\ldots, (i, x).M_i, \ldots\} \Downarrow T}$$

$$\frac{M \Downarrow \lambda x.N \qquad N[V/x] \Downarrow T}{M\ V \Downarrow T} \qquad \frac{M \Downarrow \lambda\{\ldots, i.N_i, \ldots\} \qquad N_i \Downarrow T}{M\ \hat{\imath} \Downarrow T}$$

Table 2.2: Big-step semantics for Call-By-Push-Value

2.2 Operational Semantics

Since the main purpose of CBPV is to subsume the simply-typed λ-calculus under fixed evaluation strategies, like CBN and CBV, its operational semantics also defines a fixed evaluation order. Levy defines two equivalent operational semantics, one is a big-step semantics, the other is formulated in terms of an abstract CK machine, introduced by Felleisen and Friedman in [FF86]. We will introduce the big-step semantics and give a short overview over the CK machine.

In CBPV values never need to be evaluated and are thus always terminal. For computation types, we have to define the set of terms we consider terminal, i.e. not further reducible.

Definition 2.2.1 (Terminal computations)**.** The following computations are considered to be *terminal*

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

Thus, terms are only reduced to weak head normal from, meaning that M and the M_i need not be terminal. The set of big-step reduction rules can be found in Table 2.2. The notation $M \Downarrow T$ can be read as "M reduces to terminal T with any (finite) number of steps". So, in contrast to the small-step reduction we have seen for the simply typed λ-calculus, there is no notion of a single reduction step. This will be different in the following machine semantics.

The CK machine evaluates a given computation to a terminal form by implementing the big-step rules step by step, which gives us a more fine

grained notion of a reduction step. Whenever the big-step rules require evaluation of a subterm, the context is pushed onto a stack and the subterm becomes the new term to be evaluated. If we include types, the configuration of the CK machine is a tuple $(M, \underline{B}, K, \underline{C})$ where $M : \underline{B}$ is the term currently being evaluated, K is the stack, and $\underline{C}$ is the type of the overall computation. So the last component does not change during execution. The machine name CK stands for *control string – continuation*. In our context the control string is the term M with the stack being its continuation.

The initial configuration to evaluate a term M of type $\underline{B}$ is $(M, \underline{B}, \texttt{nil}, \underline{B})$, where `nil` is the empty stack. The big-step rules can then be translated into transition rules for the machine. For example, the rule for the `to` binding:

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

can be translated as:

$$\begin{array}{llllll} & M\ \texttt{to}\ x.N & \underline{B} & & K & \underline{C} \\ \rightsquigarrow & M & F\ A & [\cdot]\ \texttt{to}\ x.N :: K & & \underline{C} \\ \\ & \texttt{return}\ V & F\ A & [\cdot]\ \texttt{to}\ x.N :: K & & \underline{C} \\ \rightsquigarrow & N[V/x] & \underline{B} & & K & \underline{C} \end{array}$$

and the rule for application:

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

can be translated into the following transition rules:

$$\begin{array}{llllll} & M\ V & \underline{B} & & K & \underline{C} \\ \rightsquigarrow & M & A \to \underline{B} & V :: K & & \underline{C} \\ \\ & \lambda x.N & A \to \underline{B} & V :: K & & \underline{C} \\ \rightsquigarrow & N[V/x] & \underline{B} & & K & \underline{C} \end{array}$$

This is where the name Call-By-Push-Value comes from. Functions are called by pushing their argument – which must always be a value – onto the stack. Then the function is evaluated until there is a λ on top and we can do substitution into the body. At this point, note that P. B. Levy uses a different notation for application to emphasize this "push and pop" reading. While he writes application operand first – $V'M$ – we use the usual notation of the λ-calculus – $M\ V$. The main reason for this is to keep notation clear in the presence of successor terms which we will usually denote with M'. Furthermore, we are heading for a nondeterministic reduction relation,

where we can reduce redexes anywhere in the term, so the "push and pop" reading does not directly apply.

2.3 A Nondeterministic Reduction Relation

For the remainder of this document, we will investigate the CBPV calculus in a slightly different setting. The original calculus is defined with a deterministic reduction relation where reductions only occur at top-level and in particular there is no reduction underneath of λ. We will now generalize the reduction relation to allow reductions at every position in a term. This of course implies that values, especially those encapsulating computations, can also be reduced.

We will base our reduction relation on the equational theory of the CBPV calculus [Lev99]. The equational theory defines provable equality between terms. In particular the equational theory respects the operational semantics from above in the sense that $M \Downarrow T$ implies that $M = T$ is provable using the equations. The equational theory is also respected by the denotational semantics of all models of CBPV. When writing down equations we use the following definition:

Definition 2.3.1. Let x be a variable and M a term. The variable x is free in M if $\Gamma = \Gamma', x : A, \Gamma''$ whenever $\Gamma \vdash^c M : \underline{B}$. We write ${}^xM$ to denote that x is not free in M. The definition for values is analog.

We derive our reduction relation from the equational theory by giving each equation a direction in which it can be applied to the term. If we consider, for example, the β-equality for `to`:

$$(\texttt{return}\ V)\ \texttt{to}\ x.M = M[V/x]$$

we transform this into the reduction rule:

$$(\texttt{return}\ V)\ \texttt{to}\ x.M \to M[V/x]$$

The set of all reduction rules can be found in Table 2.3. There are 3 classes of reduction rules. The β-reductions roughly correspond to those reductions performed by the CK machine, with the difference that in our case β-reduction can occur anywhere in the term. η-reduction is exactly the converse operation to η-expansion, which allows any term to be transformed into an introductory form of the respective type. Finally the assoc-rules allow rewriting of the stacks, we will introduce in the next section.

We denote the reduction relation with $\to$ and its reflexive, transitive closure with $\to^*$, as usual. The reduction relation has the following properties:

Proposition 2.3.2. *Reduction in Call-By-Push-Value preserves types and is preserved under substitution.*

β-reductions:

$$\begin{aligned}
(\texttt{return}\ V)\ \texttt{to}\ x.M &\longrightarrow M[V/x] \\
\texttt{force}\ \texttt{thunk}\ M &\longrightarrow M \\
\texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \{\ldots, (i,x).M_i, \ldots\} &\longrightarrow M_{\hat{\imath}}[V/x] \\
\texttt{pm}\ (V, V')\ \texttt{as}\ (x,y).M &\longrightarrow M[V/x, V'/y] \\
(\lambda\{\ldots, i.M_i, \ldots\})\ \hat{\imath} &\longrightarrow M_{\hat{\imath}} \\
(\lambda x.M)V &\longrightarrow M[V/x]
\end{aligned}$$

η-reductions:

$$\begin{aligned}
M\ \texttt{to}\ x.\texttt{return}\ x &\longrightarrow M \\
\texttt{thunk}\ \texttt{force}\ V &\longrightarrow V \\
\texttt{pm}\ V\texttt{as}\ \{\ldots, (i,x).^{x}M[(i,x)/\texttt{z}], \ldots\} &\longrightarrow M[V/z] \\
\texttt{pm}\ V\texttt{as}\ (x,y).^{xy}M[(x,y)/z] &\longrightarrow M[V/z] \\
\lambda\{\ldots, i.(M\ i), \ldots\} &\longrightarrow M \\
\lambda x.^{x}Mx &\longrightarrow M
\end{aligned}$$

assoc-rules:

$$\begin{aligned}
(M\ \texttt{to}\ x.M')\ \texttt{to}\ y.^{x}M'' &\longrightarrow M\ \texttt{to}\ x.(M'\ \texttt{to}\ y.^{x}M'') \\
(M\ \texttt{to}\ x.M')\ \hat{\imath} &\longrightarrow M\ \texttt{to}\ x.(M'\ \hat{\imath}) \\
(M\ \texttt{to}\ x.M')\ V &\longrightarrow M\ \texttt{to}\ x.(M'\ V)
\end{aligned}$$

Table 2.3: Reductions in the CBPV calculus

(i) $\Gamma \vdash^c M : \underline{B} \wedge M \to M' \Rightarrow \Gamma \vdash^c M' : \underline{B}$
$\Gamma \vdash^v V : A \wedge V \to V' \Rightarrow \Gamma \vdash^v V' : A$

(ii) $M \to M' \Rightarrow M[V/x] \to M'[V/x]$
$V \to V' \Rightarrow V[W/x] \to V'[W/x]$

Proof. By induction over the derivation of $M : A$ and the structure of M respectively. □

Proposition 2.3.3. *The reduction relation contains the big-step reductions: If $M \Downarrow T$ then $M \to^* T$ as well.*

Proof. By induction over the big-step rules using the β-reductions. □

2.4 Stacks

In conjunction with the reduction relation we just defined, we will also use the stacks from the CK machine. As we have already seen, stacks represent evaluation contexts for computation terms. The CK machine uses this to move parts of a term into its context in order to evaluate subexpressions. If we look at the second rule for `to` in more detail, we can split this up into two operations.

	$\texttt{return}\ V$	$F\ A$	$[\cdot]\ \texttt{to}\ x.N :: K$	$\underline{C}$
$\approx$	$\texttt{return}\ V\ \texttt{to}\ x.N$	$\underline{B}$	K	$\underline{C}$
$\xrightarrow{\beta}$	$N[V/x]$	$\underline{B}$	K	$\underline{C}$

The first is a rewriting of the term being evaluated and its context, "moving" a "stack frame" from the stack to the term, and the second is a β-reduction step. If we generalize this, we get an operation $\bullet$ on stacks, which is called dismantling. The details are in Table 2.4. This way we can view a computation term and an accompanying stack as a single term obtained from completely unwinding the stack. Thus, stacks and stack dismantling are, in this context, merely notation and in particular no reductions. When dealing with stacks, we also use $K +\!\!+ L$ to denote the concatenation of K and L.

So far we have no way of typing stacks. To achieve this we introduce a third typing judgement where $\Gamma \vdash^k K : \underline{B} \multimap \underline{C}$ means that K takes a computation M of type $\underline{B}$ such that $M \bullet K$ is a computation term of type $\underline{C}$. The typing rules can be found in Table 2.5. Inspection of the CK machine shows that if the machine is in configuration $(M, \underline{B}, K, \underline{C})$ the stack K must have type $\underline{B} \multimap \underline{C}$.

We have seen that once a computation of appropriate type is dismantled into a stack the construct becomes a computation term. Since there are reductions that can occur regardless of the term plugged into the stack, it is intuitive to define these as reductions on the stack itself.

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

Table 2.4: Dismantling for stacks

$$
\frac{}{\Gamma \vdash^k \mathtt{nil} : \underline{B} \multimap \underline{B}} \qquad \frac{\Gamma \vdash^k K : \underline{B} \multimap \underline{C} \quad \Gamma, x : A \vdash^c N : \underline{B}}{\Gamma \vdash^k [\cdot] \mathtt{to}\ x.N :: K : F\,A \multimap \underline{C}}
$$

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

Table 2.5: Typing rules for stacks

Definition 2.4.1 (Reductions on stacks)**.** Let K be a stack of type $\underline{B} \multimap \underline{C}$ then

$$
K \to K' \quad :\Longleftrightarrow \quad \forall M : \underline{B} \quad : \quad M \bullet K \to M \bullet K'
$$

where $M \bullet K$ denotes the term obtained by plugging M into K and unwinding the whole stack, as defined above.

Additionally, we also define the length of a stack and prove that reductions on the stack do not increase the stack length. This will be important once we prove strong normalization of CBPV.

Definition 2.4.2 (Stack length)**.** For every stack K we write $|K|$ for the length of the stack, i.e. $|\mathtt{nil}| = 0$ and $|F :: K| = 1 + |K|$ for all stack forms F.

Lemma 2.4.3 (Reductions and stack length)**.** *For every stack K we have*

$$
K \to K' \Rightarrow |K| \geq |K'|
$$

Proof. Let $K : \underline{B} \multimap \underline{C}$ be some stack and $M : \underline{B}$ with $M \bullet K \to M \bullet K'$. There are four cases, where the stack length may change. Three of them involve the assoc-rules and an interaction between two stack forms, while the fourth is the η-rule for `to`:

assoc1: If $K = L +\!\!+ [\cdot] \mathtt{to}\ x.N :: [\cdot] \mathtt{to}\ y.M :: L'$ then

$$
\begin{aligned}
M \bullet K &= M \bullet (L +\!\!+ [\cdot] \mathtt{to}\ x.N :: [\cdot] \mathtt{to}\ y.M :: L') \\
&= ((M \bullet L) \mathtt{to}\ x.N) \mathtt{to}\ y.M \bullet L' \\
&\to (M \bullet L) \mathtt{to}\ x.(N \mathtt{to}\ y.M) \bullet L' \\
&= M \bullet (L +\!\!+ [\cdot] \mathtt{to}\ x.(N \mathtt{to}\ y.M) :: L') = M \bullet K'
\end{aligned}
$$

and thus $|K'| = |K| - 1$. The remaining assoc cases can be handled by a similar use of dismantling.

assoc2:
$$\begin{array}{lll} K & = L & +\!\!+ [\cdot]\ \texttt{to}\ x.N :: \hat{\imath} \ :: L' \\ & \to L & +\!\!+ [\cdot]\ \texttt{to}\ x.N\ \hat{\imath} \ :: L' = K' \end{array}$$

assoc3:
$$\begin{array}{lll} K & = L & +\!\!+ [\cdot]\ \texttt{to}\ x.N :: V \ :: L' \\ & \to L & +\!\!+ [\cdot]\ \texttt{to}\ x.N\ V \ :: L' = K' \end{array}$$

η **:** If $K = L +\!\!+ [\cdot]\ \texttt{to}\ x.\texttt{return}\ x :: L'$ then the stack length may change, even though there is no interaction between stack forms:

$$\begin{aligned} M \bullet K &= M \bullet (L +\!\!+ [\cdot]\ \texttt{to}\ x.\texttt{return}\ x :: L') \\ &= ((M \bullet L)\ \texttt{to}\ x.\texttt{return}\ x) \bullet L' \\ &\to (M \bullet L) \bullet L') \\ &= M \bullet (L +\!\!+ L') = M \bullet K' \end{aligned}$$

In each of these cases $|K'| = |K| - 1$. We do not have to consider other reduction rules, as their prerequisites can only occur as subterms of a stack form. So whenever K reduces to K', reductions are confined to a single stack form and no stack form can disappear due to η. Thus $|K|$ does not change. □

Now that we have introduced the calculus with stacks and a new notion of reduction, we turn towards the proof theoretic aspect of strong normalization. In this context we will use the following definitions:

Definition 2.4.4 (Strong normalization)**.**

- A term M is *strongly normalizing* iff there is no infinite reduction sequence $M \to M_1 \to M_2 \to \ldots$ beginning with M. We write $M \in \mathcal{SN}$ to denote that a term is strongly normalizing.
- Similarly a stack K is strongly normalizing ($K \in \mathcal{SN}$), if there is no infinite reduction sequence starting with K

Chapter 3

Strong Normalization

In this chapter we introduce the proof techniques we will later use to prove strong normalization of the Call-By-Push-Value calculus, under the nondeterministic reduction relation we introduced in Section 2.3. First we introduce the basic technique for normalization proofs of most variants of the λ-calculus. Afterwards, we will introduce a technique that was used to prove strong normalization of Moggi's computational metalanguage λ_{ml} [Mog91, LS05].

3.1 Strong Normalization of the Simply-Typed λ-Calculus

The simply typed λ-calculus, as defined in Table 1.1, is a rewrite system for terms. One of the simplest ways to prove strong normalization of such a rewrite system is to embed the system into a terminating relation and showing for each instance of a rule $L \to R$ that (L, R) is in the relation, thus every reduction step reduces the term with respect to the terminating relation and we get strong normalization. Unfortunately, it is difficult to show this property for the rule $(\lambda x.M)\ N \to M[N/x]$ since the substitution can create any number of instances of the term N

Moreover, it is clear that the proof technique for the simply-typed λ-calculus should make use of the type system. The reason for this is that in the untyped λ-calculus we can define terms like $\Omega = (\lambda x.x\,x)\ (\lambda x.x\,x)$. This term β-reduces to itself, leading to an infinite sequence of reductions $\Omega \xrightarrow{\beta} \Omega \xrightarrow{\beta} \cdots$. However, this is not possible in the simply-typed λ-calculus, since Ω does not have a simple type.

The standard proof technique for proving strong normalization of the simply-typed λ-calculus is originally due to Tait [Tai67]. It was later simplified and adapted to system F by Girard [Gir72] and again simplified by Tait [Tai72]. A nice presentation of the proof, which we will also follow here, can be found in [GLT89, Chapter 6]. The proof uses an abstract notion of

reducibility. This reducibility relation – or predicate, since the relation is unary – is a type-indexed family of relations where each red_A is a subset of the terms of type A.

The relation we will use here is in fact a special case of the more general concept of logical relations, which are discussed for example by Mitchell [Mit96]. Such a logical relation is a family of typed relations defined by requiring some property, in our case strong normalization, at base type and then lifting this predicate uniquely to non basic types by requiring certain closure properties for each type constructor. In the case of the simply-typed λ-calculus we have the type constructors $\to$ and $\times$. So we require the relation $\text{red}_{A\to B}$ to depend on the relations red_A and red_B in a way that the family is closed under application and lambda abstraction. Similarly, we require $\text{red}_{A\times B}$ to depend on red_A and red_B such that the family is closed under pairing and projection. This leads to definitions which are inductive on the type structure of the calculus. Richer calculi, like CBPV, require additional closure properties for each type constructor, which turn out to be difficult to define inductively.

For the simply-typed λ-calculus we define the reducibility relation as follows:

- If M is of atomic type A, then $M \in \text{red}_A$ iff M is strongly normalizing.
- If M is of type $A \times B$, then $M \in \text{red}_{A\times B}$ iff $\pi_1 M \in \text{red}_A$ and $\pi_2 M \in \text{red}_B$.
- If M is of type $A \to B$ then $M \in \text{red}_{A\to B}$ iff $\forall N \in \text{red}_A : M\ N \in \text{red}_B$.

The proof then proceeds in two steps. The first step is to prove the following properties, which can be proven by induction over the structure of the type A.

(C1) If $M \in \text{red}_A$, then M is strongly normalizing.

(C2) If $M \in \text{red}_A$ and $M \to M'$, then $M' \in \text{red}_A$.

(C3) If M is not of the form (N, N') or $\lambda x.N$ and whenever we convert a redex in M we obtain a term $M' \in \text{red}_A$, then $M \in \text{red}_A$.

The next step is to show that all well-typed terms are reducible. From this we can conclude by (C1) that all terms are strongly normalizing. The proof goes by induction over the typing derivation of the calculus. This means that for each typing rule:

$$\frac{\Gamma \vdash M_1 : A_1 \qquad \ldots \qquad \Gamma \vdash M_k : A_k}{\Gamma \vdash N : B}$$

we may assume the induction hypothesis for all M_i and need to prove the claim for N. Unfortunately, this is not sufficient to directly prove that all

terms are reducible. We need to strengthen our induction hypothesis to handle the case of abstraction. Thus, we show the following:

All well-typed terms are reducible if the free variables are substituted with reducible terms.

Since the cases for the different typing rules are independent, this proof can be modularized very nicely. We will present the case of abstraction:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \tag{3.1}$$

the other cases can be found in [GLT89]. The idea is to prove the following property:

If for all $N \in \mathrm{red}_A$, $M[N/x] \in \mathrm{red}_B$ *then* $\lambda x.M \in red_{A \to B}$

The premise is exactly what we get from (3.1) and the strengthened induction hypothesis, when Γ is empty. To show $\lambda x.M \in \mathrm{red}_{A \to B}$, we need to show that for all $N \in \mathrm{red}_A$ we have $(\lambda x.M)\ N \in \mathrm{red}_B$. This can be done by showing that all successors of $(\lambda x.M)\ N$ are reducible and then applying (C3). There are three possible reductions:

- $(\lambda x.M)\ N \to M[N/x]$ which is reducible by hypothesis.
- $(\lambda x.M)\ N \to (\lambda x.M')\ N$ with $M \to M'$ or
- $(\lambda x.M)\ N \to (\lambda x.M)\ N'$ with $N \to N'$ which can both be handled by induction on the length of the longest reduction sequence of M and N respectively, as both are known to be reducible and thus strongly normalizing by (C1).

From the strengthened property above, strong normalization of the simply-typed λ-calculus can be derived by substituting the free variables with themselves using the identity substitution. This works since variables are clearly reducible by (C3). This gives us the basic technique underlying our normalization proof.

In the case of the simply-typed λ-calculus the definitions for the reducibility predicate are relatively straight forward. The reason for this is that in the λ-calculus the elimination construct for each type results in a simpler type. This allows reducibility to be easily defined by induction on the type structure. In the next section we will investigate a technique for proving strong normalization of computational calculi, which do not have this property.

Additional terms and typing rules:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash [M] : T\,A} \qquad \frac{\Gamma \vdash M : T\,A \qquad \Gamma \vdash N : T\,B}{\Gamma \vdash \texttt{let}\ x \Leftarrow M\ \texttt{in}\ N : B}$$

Additional reduction rules:

$$\begin{gathered}
\texttt{let}\ x \Leftarrow [N]\ \texttt{in}\ M \longrightarrow M[N/x] \\
\texttt{let}\ x \Leftarrow M\ \texttt{in}\ [x] \longrightarrow M \\
\texttt{let}\ y \Leftarrow (\texttt{let}\ x \Leftarrow L\ \texttt{in}\ M)\ \texttt{in}^x N \longrightarrow \texttt{let}\ x \Leftarrow L\ \texttt{in}\ (\texttt{let}\ y \Leftarrow M\ \texttt{in}\ {}^x N)
\end{gathered}$$

Table 3.1: Additional rules for λ_{ml}

3.2 $\top\top$-Lifting for Monadic Types

In the pure simply-typed λ-calculus it is very difficult to model computational effects, like printing or state. One approach to model computational effects in this setting is to extend the simply-typed λ-calculus with monadic types. Moggi's computational metalanguage λ_{ml} is such a calculus, which introduces a new type constructor T which has to be a strong monad [Mog91]. The additional terms, typing rules, and reduction rules can be found in Table 3.1.

The idea of this construction is to distinguish values of type A from computations type $T\,A$, which return values of type A, but may additionally cause effects. This way, computational effects can be absorbed into the structure of the monad. This works, because the first derivation rule, which inserts a value into the monad, is not reversible. Thus, there is no way to "leave" the monad, which would be equal to dropping the computational effects. Practical uses of this include the Haskell programming language, where the `Monad` type class is used to capture various effects in an otherwise pure language [Jon03]. In λ_{ml} the nature of the monad is intentionally left unspecified to keep the calculus as general as possible.

Unfortunately, the new derivation rules noted above make it difficult to prove strong normalization of λ_{ml} via reducibility. In the classic normalization proof, the elimination construct for each type constructor is used to define the reducibility relation by induction on the type structure. For λ_{ml}, this straightforward approach fails, since the elimination construct for $T\,A$ is not inductive on the type structure, as we can only get to terms of type $T\,B$ starting from a term of type $T\,A$.

There are, however, several ways to prove strong normalization of λ_{ml}. Benton et al. embedded the calculus into the simply-typed λ-calculus with sums and then used Prawitz's normalization result for that calculus [Pra71, BBdP98]. Another approach is due to Lindley and Stark. They work their

way to a reducibility relation by induction on the type structure and give a standalone proof in the style of Girard/Tait, which we will shortly introduce here [LS05].

The question is how to define such a reducibility relation. The naive approach could be to mimic the definition for functions.

Bad 1 $M \in \mathrm{red}_{T\,A} \iff \forall N \in \mathrm{red}_{T\,B} : \mathtt{let}\ x \Leftarrow M\ \mathtt{in}\ N \in \mathrm{red}_{T\,B}$

But this is not inductive over types as type $T\,B$ can be more complex than $T\,A$. One could try to patch this by dropping the dependence on $T\,B$

Bad 2 $M \in \mathrm{red}_{T\,A} \iff \forall N \in \mathrm{red}_{T\,B} : \mathtt{let}\ x \Leftarrow M\ \mathtt{in}\ N \in \mathcal{SN}$

But this is too weak to prove the properties of M in the presence of nested `let` constructs like $\mathtt{let}\ y \Leftarrow (\mathtt{let}\ x \Leftarrow [\cdot]\ \mathtt{in}\ N)\ \mathtt{in}\ P$. But the structure of nested `let` constructs leads to a definition of reducibility which is inductive on types. For this we need a notion of continuation in λ_{ml}, which we get from the following definitions:

- A *term abstraction* $[\cdot]\ \mathtt{to}\ x.N$ of type $T\,A \multimap T\,B$ is a computation term N of type $T\,B$ with a hole accepting terms of type $T\,A$.
- A *continuation* is a finite list of term abstractions, ended by `nil`.
- Continuations can by typed as follows:

$$\frac{}{\Gamma \vdash \mathtt{nil} : T\,A \multimap T\,A}$$

$$\frac{\Gamma \vdash [\cdot]\ \mathtt{to}\ x.N : T\,A \multimap T\,B \qquad \Gamma \vdash K : T\,B \multimap T\,C}{\Gamma \vdash ([\cdot]\ \mathtt{to}\ x.N :: K) : T\,A \multimap T\,C}$$

- A continuation of type $T\,A \multimap T\,B$ is applied to a computation of type $T\,A$ using the following rules

$$\begin{aligned} M \bullet \mathtt{nil} &= M \\ M \bullet ([\cdot]\ \mathtt{to}\ x.N :: K) &= (\mathtt{let}\ x \Leftarrow M\ \mathtt{in}\ N) \bullet K \end{aligned}$$

So the application of a term $M : T\,A$ to a continuation $K : T\,A \multimap T\,B$ can be rewritten into a term of type $T\,B$. This can easily be verified by induction on K.

The above definitions correspond directly to the stacks of Call-By-Push-Value, where $[\cdot]\ \mathtt{to}\ x.N$ is the only stack frame used, and thus, all stacks have type $F\,A \multimap F\,B$. So we can define reductions on continuations and the length for continuations as in Section 2.4. Note that in λ_{ml} we can have

a value (and variables) of type $T\,A$, which we cannot in CBPV. So $T\,A$ corresponds to $U\,(F\,A)$ rather than $F\,A$, since, from the CBPV perspective, a value of type $T\,A$ is a thunk of a computation returning a value of type A [Lev99].

Now we are in the position to define reducibility at type $T\,A$ via reducibility for continuations. For the definition of $\mathrm{red}_{T\,A}$ we need an auxiliary set of type indexed relations $\mathrm{red}^k_{T\,A}$, where each $\mathrm{red}^k_{T\,A}$ is a subset of the continuations accepting terms of type $T\,A$. With this, we get the following notions of reducibility:

- $M \in \mathrm{red}_{T\,A}$ iff for all $K \in \mathrm{red}^k_{T\,A}$ the application $M \bullet K$ is strongly normalizing.

- $K \in \mathrm{red}^k_{T\,A}$ iff for all $V \in \mathrm{red}_A$ the application $[V] \bullet K$ is strongly normalizing.

This is enough to prove strong normalization of λ_{ml}. Due to the great similarities between λ_{ml} computations and the CBPV returner type, we will not present the proof here. In particular, our Lemmas 4.3.5 and 4.3.6 are only mild extensions, to the additional types of CBPV, of the corresponding lemmas for λ_{ml}, given in [LS05].

Instead we will look at the operation of $\top\top$-lifting, defined by Lindley and Stark [LS05], as a general way to lift any predicate, not just reducibility, from a definition at value type to the corresponding monadic type. This is achieved by a "leap-frog" over continuations. Assume any observation predicate $\mathcal{O} \subseteq Terms$, and a predicate $\phi \subseteq \{V \mid \Gamma \vdash V : A\}$ on the value type A. $\top\top$-lifting can then be defined as follows:

$$\phi^{\top} = \{K \mid \forall V \in \phi : [V] \bullet K \in \mathcal{O}\}$$
$$\phi^{\top\top} = \{M \mid \forall K \in \phi^{\top} : M \bullet K \in \mathcal{O}\}$$

This way we get a predicate on the corresponding monadic type as well as the respective continuation types. In particular, the above case of reducibility is obtained by choosing $\mathcal{O} = \mathcal{SN}$. The construction is inspired by similar constructions to specific computational effects and generalizes them to some arbitrary strong monad T. Pitts and Stark, for example, use the method to define a structurally inductive notion of contextual equivalence for a subset of Standard ML with integer references [PS98].

Similar problems to those occurring in the monadic case also occur when proving strong normalization of the simply-typed λ-calculus with sums. If

we extend the calculus with the following constructs:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \iota_1 M : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \iota_2 M : A + B}$$

$$\frac{\Gamma \vdash M : A + B \qquad N_1 : T\,C \qquad N_2 : T\,C}{\texttt{case}\ M\ \texttt{of}\ \iota_1(x) \Rightarrow N_1 \mid \iota_2(x_2) \Rightarrow N_2 : T\,C}$$

we can see that the elimination rule is again not inductive on the type structure. This case can however be handled by adapting the notion of continuation to include term abstractions of type $A + B \multimap T\,C$. Here we have the restriction that both branches of the `case` construct need to be computations. This can be remedied by moving from continuations to frame stacks as in [Lin05].

It is noteworthy that the approach of $\top\top$-lifting is also strong enough to handle *commuting conversions* or *permutative conversions*, like the last reduction rule in Table 3.1. The name comes from the transforming action via the Curry-Howard isomorphism on derivation trees in natural deduction. Commuting conversions arise, for example, when one wishes to maintain the subformula property for normal proofs in the presence of disjunction and are known to cause difficulties in normalization proofs. This was originally handled by Prawitz [Pra71]. Another nice proof handling strong normalization of the simply-typed λ-calculus with sums and commuting conversions from a logical perspective can be found in [dG02]. Here the calculus is embedded into the the simply-typed λ-calculus without sums. However, the embedding is quite involved and there is no straightforward intuition given.

Chapter 4

Strong Normalization of Call-By-Push-Value

In this chapter we prove strong normalization of a slightly restricted version of the Call-By-Push-Value calculus under the nondeterministic reduction relation introduced in Section 2.3. So before starting with the proof, we show why it is necessary to restrict the calculus. We then introduce a reducibility relation, defined by induction on the types of CBPV. For this we define a $\top\top$-*closure* operation, closely related to the approach of $\top\top$-lifting.

4.1 Restriction

There is one restriction we have to make to CBPV, in order to actually make the calculus strongly normalizing. CBPV provides generalized sum and product types, $\sum_{i \in I} A_i$ and $\prod_{i \in I} \underline{B}_i$, where the only restriction on the index set I is to be countable. This leads to an infinitely wide syntax which is not strongly normalizing. Consider the following term Θ of type $\prod_{i \in \mathbb{N}} F\, 1$:

$$\lambda\{\ldots, i.((\lambda x.\texttt{return}\ x)\ ()), \ldots\}$$

There are infinitely many subterms of the form $(\lambda x.\texttt{return}\ x)()$ and each can make a single β-reduction. So we get an infinite sequence of reductions starting at Θ. For sum types the values of a type like `nat` themselves do not present a problem. Instead the corresponding pattern match construct $\texttt{pm}\ V\ \texttt{as}\ \{\ldots, (i, x).M_i, \ldots\}$ becomes infinitely wide. So in the following we restrict ourselves to the case of finite index sets I for sums and products.

This of course means that types like `nat` can no longer be defined within the language and need to be added externally. However, types with finitely many elements, like `bool`, can still be defined within the language. So for practical purposes, like considering CBPV as a meta language in a compiler, this restriction is irrelevant.

Another approach could be to define a notion of parallel reductions on infinitely wide terms by reducing all terms underneath of $\lambda\{\ldots\}$ simultaneously, as long as they still contain redexes. However, this would completely change our model of reduction, so we are not investigating this idea further. On the other hand, the non-normalization of full CBPV is completely different from the non-normalization of the untyped λ-calculus. There are no terms like Ω in CBPV that reduce to themselves. With this restriction the calculus becomes finitely branching and we can define:

Definition 4.1.1. If $P \in \mathcal{SN}$ we write $\max(P)$ for the length of a longest reduction sequence beginning with P.

4.2 Defining Reducibility

In section 3.2 we have seen how $\top\top$-lifting can be used to define reducibility by induction on types for the computational metalanguage λ_{ml}. In this section we will adapt this technique to the types of CBPV to get a uniform treatment of all the computation types. For this we use an operation we call $\top\top$-closure, which roughly corresponds to the unary case of the relational $\top\top$-closure used by Pitts to define a form of observational congruence for PolyPCF [Pit00].

The problem we get when trying to apply $\top\top$-lifting directly to CBPV is that the definition of the lifting already includes the term construct for turning a value V into a computation, namely $[V]$ for λ_{ml} or `return` V the CBPV analog. So to reach a uniform treatment of all computation types, we have to factor this construction out of the lifting operation. So instead of lifting a predicate from a value type to a computation type, we aim to define a part of the predicate already at computation type and then use a suitable closure operator. We do this by defining the following symmetric operations:

Definition 4.2.1 ($\top\top$-closure)**.**

- Let $Terms(\underline{B})$ be the set of all terms M of type $\underline{B}$ and let $Stacks(\underline{B})$ be the set of all stacks of type $\underline{B} \multimap \underline{C}$ for any computation types $\underline{B}$ and $\underline{C}$
- We define the operation $(-)^\top : \mathcal{P}(Terms(\underline{B})) \to \mathcal{P}(Stacks(\underline{B}))$

$$T^\top = \{K \mid \forall M \in T : M \bullet K \in \mathcal{SN}\}$$

- And in the other direction $(-)^\top : \mathcal{P}(Stacks(\underline{B})) \to \mathcal{P}(Terms(\underline{B}))$

$$S^\top = \{M \mid \forall K \in S : M \bullet K \in \mathcal{SN}\}$$

Lemma 4.2.2. *The operations* $(-)^\top$ *form an antitone Galois connection with respect to inclusion:*

$$T \subseteq S^\top \iff S \subseteq T^\top \qquad (*)$$

and thus $(-)^\top$ *is:*

inclusion reversing: $A \subseteq B \Rightarrow B^\top \subseteq A^\top$

and $(-)^{\top\top}$ *is:*

monotone: $A \subseteq B \Rightarrow A^{\top\top} \subseteq B^{\top\top}$
inflationary: $A \subseteq A^{\top\top}$
idempotent: $A^{\top\top} = (A^{\top\top})^{\top\top}$

Proof. Using Definition 4.2.1 we can rewrite $S \subseteq T^\top$ as:

$$\begin{aligned} S \subseteq T^\top &\iff \forall K : (K \in S \Rightarrow \forall M \in T : M \bullet K \in \mathcal{SN}) \\ &\iff \forall K \in S : \forall M \in T : M \bullet K \in \mathcal{SN} \\ &\iff \forall M \in T : \forall K \in S : M \bullet K \in \mathcal{SN} \iff T \subseteq S^\top \end{aligned}$$

We then get the remaining properties from $(*)$. We prove the properties for $T, T' \subseteq Terms(\underline{B})$. The case for stacks is completely symmetric.

inflationary: Since for any set we have $T^\top \subseteq T^\top$ we get $T \subseteq T^{\top\top}$ using $(*)$ with $S = T^\top$.

order reversing: Assume $T \subseteq T'$ then $T \subseteq T' \subseteq (T')^{\top\top}$ and we get $(T')^\top \subseteq T^\top$ using $(*)$ with $S = (T')^\top$

monotone: Since $(-)^\top$ is order reversing in both directions it follows immediately that $(-)^{\top\top}$ is monotone.

idempotent: Since $(-)^{\top\top}$ is inflationary is sufficient to prove $(T^{\top\top})^{\top\top} \subseteq T^{\top\top}$. Applying $(*)$ to $T^{\top\top} \subseteq T^{\top\top}$ yields $T^\top \subseteq (T^{\top\top})^\top$. Thus, we get $(T^{\top\top})^{\top\top} \subseteq T^{\top\top}$ by applying $(-)^\top$ which is order reversing.

□

Thus, $(-)^{\top\top}$ is a closure operation. We can use this closure operation to define reducibility for the computation types of CBPV by induction on the type structure. We do this by choosing a base set having a particular syntactic structure, in our case the introductory form for the respective computation type. Such a set is easily definable by induction on types, but would be to weak to prove strong normalization. We then apply our $\top\top$-closure to obtain the full reducibility predicate on the computation type, which is strong enough to carry out subsequent proofs.

Since CBPV clearly separates computations and values using two different type judgements, we also use two different sets of reducibility relations: $\{\mathrm{red}^v_A \mid A \text{ is a value type}\}$ for value types and $\{\mathrm{red}^c_{\underline{B}} \mid \underline{B} \text{ is a computation type}\}$. Additionally we will use $\mathrm{red}^k_{\underline{B}}$ do denote reducibility for stacks accepting computations of type $\underline{B}$. Using these conventions, reducibility for computation types can be defined like this:

Definition 4.2.3 (Reducibility)**.**

- $\mathrm{red}^c_{F\,A} = \{\mathtt{return}\ V \mid V \in \mathrm{red}^v_A\}^{\top\top}$
- $\mathrm{red}^c_{A\to\underline{B}} = \{\lambda x.M \mid \forall V \in \mathrm{red}^v_A : M[V/x] \in \mathrm{red}^c_{\underline{B}}\}^{\top\top}$
- $\mathrm{red}^c_{\prod_{i\in I}\underline{B}_i} = \{\lambda\{\ldots, i.M_i, \ldots\} \mid \forall i \in I : M_i \in \mathrm{red}^c_{\underline{B}_i}\}^{\top\top}$

What remains to be done is defining reducibility for value types. Here we make use of the fact that the version of CBPV we are investigating does not include "complex values". This means that we can rely on a value being either a variable or in the introductory form of the corresponding type. For example $V : A \times A'$ is either a variable or of the form (W, W'). This allows reducibility on value types to be defined by the same construction we use to construct the base sets for computation types, with the difference that we do not need a lifting operation for values. The definitions become even simper when we exclude variables from the set of reducible terms. This spares us some case distinctions in the proof. While at first it appears that this restricts the normalization result to closed terms we will see, at the end of the normalization proof, that this is not the case. Table 4.1 contains the full definitions for red^v,red^c, and red^k expanded in the way they are applied in the normalization proof.

Strictly speaking, the closure approach may not be necessary for the types $A \to \underline{B}$ and $\prod_{i\in I}\underline{B}_i$ since we have an elimination construct for these types. Our approach, on the other hand, gives us a very uniform treatment of all computation types, which is captured by the following observation:

Observation 4.2.4. For every computation type $\underline{B}$ we have

$$\mathrm{red}^c_{\underline{B}} = \{M : \underline{B} \mid \forall K \in \mathrm{red}^k_{\underline{B}} : M \bullet K \in \mathcal{SN}\}$$

This allows some cases of the following normalization proof to be proven together.

4.3 Proof of Strong Normalization

We begin by proving the following lemma:

Reducibility for value types:

$$\begin{aligned}
\mathrm{red}^v_1 &= \{()\} \\
\mathrm{red}^v_{A\times A'} &= \{(V,V') \mid V \in \mathrm{red}^v_A, \mathrm{red}^v_{A'}\} \\
\mathrm{red}^v_{\sum_{i\in I} A_i} &= \{(i, A_i) \mid A_i \in \mathrm{red}^v_{A_i}\} \\
\mathrm{red}^v_{U\ \underline{B}} &= \{\mathtt{thunk}\ M \mid M \in \mathrm{red}^c_{\underline{B}}\}
\end{aligned}$$

Reducibility for computation types:

$$\begin{aligned}
\mathrm{red}^c_{F\,A} &= \{\mathtt{return}\ V \mid V \in \mathrm{red}^v_A\}^{\top\top} \\
&= \{M : F\,A \mid \forall K \in \mathrm{red}^k_{F\,A} : M \bullet K \in \mathcal{SN}\} \\
\mathrm{red}^c_{A\to\underline{B}} &= \{\lambda x.M \mid \forall V \in \mathrm{red}^v_A : M[V/x] \in \mathrm{red}^c_{\underline{B}}\}^{\top\top} \\
&= \{M : A \to \underline{B} \mid \forall K \in \mathrm{red}^k_{A\to\underline{B}} : M \bullet K \in \mathcal{SN}\} \\
\mathrm{red}^c_{\prod_{i\in I}\underline{B}_i} &= \{\lambda\{\ldots, i.M_i, \ldots\} \mid \forall i \in I : M_i \in \mathrm{red}^c_{\underline{B}_i}\}^{\top\top} \\
&= \{M : \Pi_{i\in I}\underline{B}_i \mid \forall K \in \mathrm{red}^k_{\prod_{i\in I}\underline{B}_i} : M \bullet K \in \mathcal{SN}\}
\end{aligned}$$

Reducibility for stacks

$$\begin{aligned}
\mathrm{red}^k_{F\,A} &= \{K : F\,A \multimap \underline{B} \mid \\
&\qquad \forall M \in \{\mathtt{return}\ V \mid V \in \mathrm{red}^v_A\} : M \bullet K \in \mathcal{SN}\} \\
\mathrm{red}^k_{A\to\underline{B}} &= \{K : (A \to \underline{B}) \multimap \underline{C} \mid \\
&\qquad \forall M \in \{\lambda x.M \mid \forall V \in \mathrm{red}^v_A : M[V/x] \in \mathrm{red}^c_{\underline{B}}\} : M \bullet K \in \mathcal{SN}\} \\
\mathrm{red}^k_{\prod_{i\in I}\underline{B}_i} &= \{K : \Pi_{i\in I}\underline{B}_i \multimap \underline{C} \mid \\
&\qquad \forall M \in \{\lambda\{\ldots, i.M_i, \ldots\} \mid \forall i \in I : M_i \in \mathrm{red}^c_{\underline{B}_i}\} : M \bullet K \in \mathcal{SN}\}
\end{aligned}$$

Table 4.1: Reducibility for the types of CBPV

Lemma 4.3.1. $M[V/x] \in \mathcal{SN} \Rightarrow M \in \mathcal{SN}$

Proof. By contraposition. Assume $M \notin \mathcal{SN}$. In this case, there is some infinite reduction sequence beginning with M. By Proposition 2.3.2 there is also a corresponding infinite reduction sequence beginning with $M[V/x]$ and thus $M[V/x] \notin \mathcal{SN}$. □

Since CBPV does not provide variables of computation type, we also need the following technical lemma:

Lemma 4.3.2 (Existence Lemma)**.** *For every computation type $\underline{B}$ there exists a closed term $M \in \mathrm{red}^c_{\underline{B}}$ and for every value type A there exists a term $V \in red^v_A$*

Proof. The term $()$ is reducible. The rest follows by induction on the type structure using the introduction rule for each type. □

Now we are in the position to prove that the reducibility predicate implies strong normalization, is preserved under reduction, and that *neutral* terms reducing only to reducible terms are also reducible, where the technical notion of *neutrality* is defined as follows:

Definition 4.3.3 (Neutral terms)**.** A term is *neutral*, if it is not of one of the following forms:

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

Note, that our notion of neutrality differs from that used in the classic normalization proof of Girard. We only need to exclude the introduction forms for computations. This is due to fact that we do not consider complex values in CBPV. Additionally, we have to exclude variables since we decided to exclude them from the set of reducible terms. Thus we can prove:

Lemma 4.3.4. *The reducibility relation for computation types fulfills the following properties:*

(C1) $M \in red^c_{\underline{B}} \Rightarrow M \in \mathcal{SN}$

(C2) $M \in red^c_{\underline{B}} \wedge M \to M' \Rightarrow M' \in red^c_{\underline{B}}$

(C3) $M : \underline{B}$ neutral $\wedge\ (\forall M' : M \to M' \Rightarrow M' \in red^c_{\underline{B}}) \Rightarrow M \in red^c_{\underline{B}}$

The relation for value types is analog.

(C1) $V \in red^v_A \Rightarrow V \in \mathcal{SN}$

(C2) $V \in red^v_A \wedge V \to V' \Rightarrow V' \in red^v_A$

(C3) $V : A$ neutral $\wedge\ (\forall V' : V \to V' \Rightarrow V' \in red^v_A) \Rightarrow V \in red^v_V$

Proof. Proof over the type structure. Since we do not consider complex values the part for value types is relatively straightforward. Furthermore, variables are neither reducible nor neutral so we only have to consider closed terms.

1: The unit type clearly satisfies all conditions, since the only closed value of type 1 is () which is reducible and normal.

$A \times A'$:

(C1) Let $W \in \mathrm{red}^v_{A\times A'}$. Thus $W = (U, V)$ with $U \in \mathrm{red}^v_A$ and $V \in \mathrm{red}^v_{A'}$ by the definition of reducibility. U and V are strongly normalizing by induction hypothesis (C1) on A and A' respectively and we have $\max(W) \leq \max(U) + \max(V)$. Thus W is strongly normalizing as well.

(C2) Let $W = (U, V)$ as above, then $W \to W'$ implies $U \to U'$ or $V \to V'$. In in first case we have $U' \in \mathrm{red}^v_A$ by induction hypothesis (C3) and thus $W' \in \mathrm{red}^v_{A\times A'}$. The other case is symmetric.

(C3) Let $W : A \times A'$ be neutral. Then $W = (U, V)$ and reduction can either happen in U or in V. In the first case $W \to (U', V) \in \mathrm{red}^v_{A\times A'}$ and $U \to U' \in \mathrm{red}^v_A$ by definition of $\mathrm{red}^v_{A\times A'}$. By induction hypothesis (C3) for A we know that $U \in red^v_A$. The same argument holds for V so $W \in \mathrm{red}^v_{A\times A'}$.

$\sum_{i\in I} A_i$:

(C1) Let $W \in \mathrm{red}^v_{\sum_{i\in I} A_i}$. Then $W = (\hat{\imath}, V)$ with $V \in \mathrm{red}^v_{A_{\hat{\imath}}}$ by definition. Since V is strongly normalizing by induction hypothesis and $W = (\hat{\imath}, V) \to W' = (\hat{\imath}, V')$ iff $V \to V'$ we have $\max((\hat{\imath}, V)) = \max(V)$ and thus W is also strongly normalizing.

(C2+C3) Follow by the same equivalence, and the respective inductions hypothesis for A_i.

$U\,\underline{B}$:

(C1) Let $W = \mathtt{thunk}\ M \in \mathrm{red}^v_{U\,\underline{B}}$ By induction hypothesis $M \in \mathcal{SN}$. We show $\mathtt{thunk}\ M \in \mathcal{SN}$ by induction on $\max(M)$. The term $\mathtt{thunk}\ M$ can reduce as follows:
If $M \neq \mathtt{force}\ V$ then $\mathtt{thunk}\ M \to \mathtt{thunk}\ M'$ with $M \to M'$ and $\max(M') < \max(M)$ only and we can apply the induction hypothesis.
If $M = \mathtt{force}\ V$ for some V of type $U\,\underline{B}$, this V must be of the

form $\texttt{thunk}\ N$ for some $N \in \text{red}^c_{\underline{B}}$ because V cannot be a variable and

$$\begin{aligned}\texttt{force}\ (\texttt{thunk}\ N) \in \text{red}^c_{\underline{B}} &\Rightarrow \forall K \in \text{red}^k_{\underline{B}} : \texttt{force thunk}\ N \bullet K \in SN \\ &\overset{\beta}{\Rightarrow} \forall K \in \text{red}^k_{\underline{B}} : N \bullet K \in SN \\ &\Rightarrow N \in \text{red}^c_{\underline{B}}\end{aligned}$$

Now $\texttt{thunk}\ M$ can also reduce as follows:

$$\begin{aligned}&\texttt{thunk}\ M \to \texttt{thunk}\ M' \text{ with } M \to M' \text{ as above} \qquad \text{or} \\ &\texttt{thunk}\ M = \underline{\texttt{thunk}\ (\texttt{force}} \ \underline{\texttt{thunk}\ N}) \\ &\qquad \overset{\eta}{\to} \texttt{thunk}\ N\end{aligned}$$

But since $M = \texttt{force thunk}\ N$ and thus $M \to N$ we have that $max(N) < max(M)$. and we can again apply the induction hypothesis.

(C2) Let $W = \texttt{thunk}\ M \in \text{red}^v_{U\,\underline{B}}$ as above. There are two cases to consider. If $M \neq \texttt{force}\ V$ then $W \to W'$ iff $M \to M'$ where $M' \in red^c_{\underline{B}}$ by induction hypothesis on $\underline{B}$ and thus $W' \in \text{red}^v_{U\,\underline{B}}$. If $M = \texttt{force}\ V$ for some V then V must be of the form $\texttt{thunk}\ N$ for some N. Thus $W = \texttt{thunk}\ (\texttt{force thunk}\ N)$ where $N \in \text{red}^c_{\underline{B}}$ by the same argument as above. W only reduces to the following reducible terms:

$$\begin{aligned}&\texttt{thunk}\ M = \underline{\texttt{thunk}\ (\texttt{force}\ \texttt{thunk}\ N)} \overset{\eta}{\to} \texttt{thunk}\ N \in \text{red}^v_{U\,\underline{B}} \qquad \text{or} \\ &\texttt{thunk}\ M \to \texttt{thunk}\ M' \in \text{red}^v_{U\,\underline{B}} \text{ with } M \to M' \in \text{red}^c_{\underline{B}}\end{aligned}$$

(C3) Let $W = \texttt{thunk}\ M$ be some term of type $U\,\underline{B}$. and $\forall W' : W \to W' \Rightarrow W' \in \text{red}^v_{U\,\underline{B}}$ By definition of $\text{red}^v_{U\,\underline{B}}$ all W' must be of the form $\texttt{thunk}\ N$ with $N \in \text{red}^c_{\underline{B}}$. In particular this holds for all reductions confined to M. By induction hypothesis on $\underline{B}$ we have $M \in \text{red}^c_{\underline{B}}$ and thus $W \in red^v_{U\,\underline{B}}$

For computation types Observation 4.2.4 allows (C2) and (C3) to be proven simultaneously:

(C2) Let $M \in \text{red}^c_{\underline{B}}$ and $M \to M'$. Since M is reducible we have $\forall K \in \text{red}^k_{\underline{B}} : M \bullet K \in \mathcal{SN}$. Since $M \to M'$ we have that $M \bullet K \to M' \bullet K$ and therefore $\forall K \in \text{red}^k_{\underline{B}} : M' \bullet K \in \mathcal{SN}$ as well. Thus $M' \in \text{red}^c_{\underline{B}}$

(C3) Let $M : \underline{B}$ be neutral and $\forall M' : M \to M' \Rightarrow M' \in \mathrm{red}^c_{\underline{B}}$. We show $\forall K \in \mathrm{red}^k_{\underline{B}} : M \bullet K \in \mathcal{SN}$.
Let $K \in \mathrm{red}^k_{\underline{B}}$. We first show that K is strongly normalizing. By Lemma 4.3.2 there exists some term $N \in red^c_{\underline{B}}$ and for this we have $N \bullet K \in \mathcal{SN}$. Since $K \to K' \iff \forall N : N \bullet K \to N \bullet K'$ we know that K is strongly normalizing. Now we can prove $M \bullet K \in \mathcal{SN}$ by induction on $\max(K)$.
$M \bullet K$ reduces to the following terms:

- $M' \bullet K$ which is strongly normalizing as $M' \in \mathrm{red}^c_{\underline{B}}$ and $K \in \mathrm{red}^k_{\underline{B}}$
- $M \bullet K'$ which is strongly normalizing by induction hypothesis

Because M is neutral and neutral terms do not interact with stack forms there are no other reductions.

The idea for proving (C1) for some type $\underline{B}$ is to show that $\texttt{nil} \in \mathrm{red}^k_{\underline{B}}$, for then we have:

$$M \in \mathrm{red}^c_{\underline{B}} \Rightarrow M \bullet \texttt{nil} \in \mathcal{SN} \Rightarrow M \in \mathcal{SN}$$

($F\,\underline{A}$) To show $\texttt{nil} \in \mathrm{red}^k_{F\,\underline{A}}$, we have to show for all M from the set $\{\texttt{return}\ V \mid V \in \mathrm{red}^v_A\}$ that $M \bullet \texttt{nil} = M \in \mathcal{SN}$. But by induction hypothesis $V \in \mathrm{red}^v_A$ implies $V \in \mathcal{SN}$ and so $\texttt{return}\ V \in \mathcal{SN}$ as well.

($A \to \underline{B}$) To show $\texttt{nil} \in \mathrm{red}^k_{F\,\underline{A}}$, let $M \in \{\lambda x.N \mid \forall V \in red^v_A : N[V/x] \in \mathrm{red}^c_{\underline{B}}\}$, and show $M \in \mathcal{SN}$ as above. If $N[V/x] \in \mathrm{red}^c_{\underline{B}}$ then $N[V/x] \in \mathcal{SN}$ by induction hypothesis. By Lemma 4.3.1 $N \in \mathcal{SN}$ and thus $\lambda x.N \in \mathcal{SN}$ The last step follows since $\lambda x.N$ can at most make one additional η-reduction if $N \to^* {}^x N'\ x$.

($\prod_{i \in I} \underline{B}_i$) Let $M \in \{\lambda\{\ldots, i.N_i, \ldots\} \mid \forall i \in I : N_i \in \mathrm{red}^c_{\underline{B}_i}\}$ and show $M \in \mathcal{SN}$. As above, the induction hypothesis on the $\underline{B}_i$ gives us $\forall i \in I : N_i \in \mathcal{SN}$ and so $\lambda\{\ldots, i.N_i, \ldots\} \in \mathcal{SN}$ as well. Thus $\texttt{nil} \in \mathrm{red}^k_{\prod_{i \in I} \underline{B}_i}$

Thus we have $\texttt{nil} \in \mathrm{red}^k_{\underline{B}}$ for any computation type $\underline{B}$. This finishes the proof. □

The next step is to prove that all terms are in fact reducible. We are heading for a proof over the typing derivation. To modularize the proof we first show some lemmas where reducibility is preserved by the typing derivation.

Lemma 4.3.5. *If $P : A$, $P \in \mathcal{SN}$, $x : A \vdash^c N : \underline{B}$, and $N[P/x] \bullet K \in \mathcal{SN}$ then $(\texttt{return}\ P\ \texttt{to}\ x.N) \bullet K \in \mathcal{SN}$*

Proof. By induction on $max(P) + max(N \bullet K) + |K|$. The term $M = (\texttt{return}\ P\ \texttt{to}\ x.N) \bullet K$ can be reduced in the following ways:

- $M \xrightarrow{\beta} N[P/x] \bullet K$ which is strongly normalizing by hypothesis

- If $N = \texttt{return}\ x$
  $M \xrightarrow{\eta} \texttt{return}\ P \bullet K = N[P/x] \bullet K \in \mathcal{SN}$ as above.

- If $K = [\cdot]\ \texttt{to}\ y.^x M :: K'$
  $M \xrightarrow{assoc1} \texttt{return}\ P\ \texttt{to}\ x.(N\ \texttt{to}\ y.^x M) \bullet K'$
  We want to apply the induction hypothesis with $N^* = N\ \texttt{to}\ y.^x M$ and $K^* = K'$. So we have to show that $N^*[P/x] \bullet K^*$ is strongly normalizing. But this is clear since:

$$\begin{aligned} N^*[P/x] \bullet K^* &= (N\ \texttt{to}\ y.^x M)[P/x] \bullet K' \\ &= N[P/x]\ \texttt{to}\ y.^x M \bullet K' \\ &= N[P/x] \bullet K \in \mathcal{SN} \end{aligned}$$

  Further we have $|K^*| < |K|$ and $N^* \bullet K^* = N \bullet K$ which implies $max(N^* \bullet K^*) = max(N \bullet K)$ so we can apply the induction hypothesis. This gives us:
  $(\texttt{return}\ P\ \texttt{to}\ x.N^*) \bullet K^* = (\texttt{return}\ P\ \texttt{to}\ x.(N\ \texttt{to}\ y.^x M)) \bullet K' \in \mathcal{SN}$ as required.

- If $K = \hat{\imath} :: K'$
  $M \xrightarrow{assoc2} (\texttt{return}\ P\ \texttt{to}\ x.N\ \hat{\imath}) \bullet K'$ We again go for the induction hypothesis with $N^* = N\ \hat{\imath}$ and $K^* = K'$ and have

$$\begin{aligned} N^*[P/x] \bullet K^* &= (N[P/x])\ \hat{\imath} \bullet K' \\ &= N[P/x] \bullet K \in \mathcal{SN} \end{aligned}$$

  Thus we can again apply the induction hypothesis which gives us
  $\texttt{return}\ P\ \texttt{to}\ x.N^* \bullet K^* = (\texttt{return}\ P\ \texttt{to}\ x.N\ \hat{\imath}) \bullet K' \in \mathcal{SN}$

- If $K = V :: K'$
  $M \xrightarrow{assoc3} (\texttt{return}\ P\ \texttt{to}\ x.N\ V) \bullet K'$ but this case is identical to *assoc2* if one substitutes V for $\hat{\imath}$.
  All other reductions are confined to either P, K or N. This decreases either $max(P)$ or $max(N \bullet K)$. By lemma 2.4.3, $|K|$ does not increase, so these cases can all be handled by the induction hypothesis, which finishes the proof.

□

Lemma 4.3.6 (Preservation – `to`)**.**
If $M \in \mathrm{red}^c_{F\,A}$, $x : A \vdash^c N : \underline{B}$, *and* $\forall P \in \mathrm{red}^v_A : N[P/x] \in \mathrm{red}^c_{\underline{B}}$ *then* $M\ \texttt{to}\ x.N \in \mathrm{red}^c_{\underline{B}}$

Proof. Let $K \in \text{red}^k_{\underline{B}}$. We then need to show that $(M \texttt{ to } x.N) \bullet K$ is strongly normalizing. Let $P \in \text{red}^v_A$. P is strongly normalizing by (C1) and since $N[P/x] \in \text{red}^c_{\underline{B}}$ we have $N[P/x] \in \mathcal{SN}$ as well.
By Lemma 4.3.5 we know that $(\texttt{return } P \texttt{ to } x.N) \bullet K$ is strongly normalizing. But this term is equal to $(\texttt{return } P) \bullet ([\cdot] \texttt{ to } x.N :: K)$. Since P was arbitrary, $([\cdot] \texttt{ to } x.N :: K) \in \text{red}^k_{F\,A}$ by definition and since $M \in \text{red}^c_{F\,A}$ we have:
$M \bullet ([\cdot] \texttt{ to } x.N :: K) = (M \texttt{ to } x.N) \bullet K \in \mathcal{SN}$. □

Lemma 4.3.7 (Preservation – `force`)**.** *If* $V \in \text{red}^v_{U\,\underline{B}}$ *then* $\texttt{force } V \in \text{red}^c_{\underline{B}}$

Proof. Let $V \in \text{red}^v_{U\,\underline{B}}$, then $V = \texttt{thunk } M$ for some $M \in red^c_{\underline{B}}$. Furthermore V is strongly normalizing by (C1). We prove the claim by induction on $max(V)$. The term $\texttt{force } V = \texttt{force thunk } M$ reduces as follows.

- $\xrightarrow{\beta} M \in \text{red}^c_{\underline{B}}$
- $\longrightarrow \texttt{force } V'$ for some reduction $V \to V'$ where $V' \in red^v_{U\,\underline{B}}$ by (C2) and $max(V') < max(V)$ so that we can apply the induction hypothesis.

□

Lemma 4.3.8. *If* $K \in \text{red}^k_{\underline{B}}$ *and* $V \in \text{red}^v_A$ *then* $V :: K \in \text{red}^k_{A \to \underline{B}}$

Proof. Let K and V be reducible as in the premise and show:

$$\forall M \in \{\lambda x.N \mid N[V/x] \in \text{red}^c_{\underline{B}}\} : M \bullet V :: K \in \mathcal{SN}$$

Since all M from this set are reducible, and thus normalizing, we can prove the claim by induction on $max(M) + max(V) + max(K)$. The only case we can not handle by induction hypothesis is the beta reduction to $N[V/x] \bullet K$. But this term is strongly normalizing since both $N[V/x]$ and K are reducible. □

Lemma 4.3.9 (Preservation – Application)**.** *If* $M \in \text{red}^c_{A \to \underline{B}}$ *and* $V \in \text{red}^v_A$ *then* $M\ V \in \text{red}^c_{\underline{B}}$

Proof. Let $K \in \text{red}^k_{\underline{B}}$. We then have to show that $(M\ V) \bullet K \in \mathcal{SN}$ By Lemma 4.3.8, $V :: K \in \text{red}^k_{A \to \underline{B}}$.Hence $M \bullet (V :: K) \in \mathcal{SN}$, since $M \in \text{red}^c_{A \to \underline{B}}$. But by the definition of $\bullet$ this is the same as $(M\ V) \bullet K$

□

Lemma 4.3.10. *If* $K \in \text{red}^k_{\underline{B}_{\hat{\imath}}}$ *and* $\hat{\imath} \in I$ *then* $\hat{\imath} :: K \in \text{red}^k_{\prod_{i \in I} \underline{B}_i}$

Proof. As for the function type stacks we prove this by showing:

$$\forall M \in \{\lambda\{\ldots, i.M_i, \ldots\} \mid \forall i \in I : M_i \in \mathrm{red}^c_{\underline{B}_i}\} : M \bullet \hat{\imath} :: K \in \mathcal{SN}$$

We take some $M = \lambda\{\ldots, i.M_i, \ldots\}$ and show that $M \bullet i :: K \in \mathcal{SN}$ by induction on $max(K) + \sum_{i \in I} max(M_i)$. All reductions in M or K can be handled by the induction hypothesis. So the only interesting case is the β-reduction to $M_{\hat{\imath}} \bullet K$. But this term is strongly normalizing since $M_{\hat{\imath}} \in \mathrm{red}^c_{\underline{B}_{\hat{\imath}}}$ and $K \in \mathrm{red}^k_{\underline{B}_i}$. □

Lemma 4.3.11 (Preservation – Projection)**.** *If* $M \in \mathrm{red}^c_{\prod_{i \in I} \underline{B}_i}$ *and* $\hat{\imath} \in I$ *then* $M\ \hat{\imath} \in \mathrm{red}^c_{\underline{B}_{\hat{\imath}}}$

Proof. The proof is similar to the function case. Let $K \in red^k_{\underline{B}_{\hat{\imath}}}$. We then need to show $(M\ \hat{\imath}) \bullet K \in \mathcal{SN}$. By Lemma 4.3.10, $\hat{\imath} :: K \in \mathrm{red}^k_{\prod_{i \in I} \underline{B}_i}$ Therefore, and since $M \in \mathrm{red}^c_{\prod_{i \in I} \underline{B}_i}$, we have $M \bullet (\hat{\imath} :: K) \in \mathcal{SN}$. But this term is equal to $(M\ \hat{\imath}) \bullet K$.

□

Lemma 4.3.12 (Preservation – $\mathtt{pm}\times$)**.** *If* $V \in \mathrm{red}^v_{A \times A'}$ *and* $\forall W \in \mathrm{red}^v_A, W' \in \mathrm{red}^v_{A'}$:
$M[W/x, W'/y] \in \mathrm{red}^c_B$ *then* $\mathtt{pm}\ V\ \mathtt{as}\ (x, y).M \in \mathrm{red}^c_B$

Proof. Since V is reducible $V = (W, W')$ for some $W \in \mathrm{red}^v_A$ and $W' \in \mathrm{red}^v_{A'}$. So it suffices to show $M_0 = \mathtt{pm}\ (W, W')\ \mathtt{as}\ (x, y).M \in \mathrm{red}^c_{\underline{B}}$. By (C1) $M[W/x, W'/y] \in \mathcal{SN}$ and thus by Proposition 4.3.1 $M \in \mathcal{SN}$. Thus we can prove by induction on $max(M) + max(V)$ that M_0 only reduces to reducible terms. Possible reductions are:

- $M_0 \xrightarrow{\beta} M[W/x, W'/y]$ which is reducible by hypothesis.
- If $M = {}^{xy}N[(x, y)/z]$ then $M_0 = \mathtt{pm}\ (W, W')\ \mathtt{as}\ (x, y).{}^{xy}N[(x, y)/z]$ and $M_0 \xrightarrow{\eta} ({}^{xy}N[(x, y)/z])[(W, W')/z] = {}^{xy}N[(W, W')/z] = M[W/x, W'/y]$ which is again reducible.
- Some reduction within W, W' or M which can be handled by (C2) and the induction hypothesis.

There are no other reductions and thus by (C3) $\mathtt{pm}\ (W, W')\ \mathtt{as}\ (x, y).M \in \mathrm{red}^c_{\underline{B}}$ □

Lemma 4.3.13 (Preservation – $\mathtt{pm}\Sigma$)**.** *If* $(\hat{\imath}, V) \in \mathrm{red}^v_{\sum_{i \in I} A_i}$ *and* $\forall i \in I\ \forall V' \in \mathrm{red}^v_{A_i} : M_i[V'/x] \in \mathrm{red}^c_{\underline{B}}$ *then* $\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots, (i, x).M_i, \ldots\} \in \mathrm{red}^c_{\underline{B}}$

Proof. We show by induction on $max(V) + \sum_{i \in I} max(M_i)$ that $M_0 = \texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \{\ldots, (i,x).M_i, \ldots\}$ only reduces to reducible terms. The term reduces as follows:

- $M_0 \xrightarrow{\beta} M_{\hat{\imath}}[V/x]$ which is reducible by hypothesis.
- If all M_i are of the form $M_i = {}^{x}N[(i,x)/z]$ for some fixed N, then: $M_0 = \texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \{\ldots, (i,x).{}^{x}N[(i,x)/z], \ldots\}$ $\xrightarrow{\eta} ({}^{x}N[(i,x)/z])[(\hat{\imath}, V)/z] = N[V/x]$ for the N at position $\hat{\imath}$ which is reducible by hypothesis.
- Some reduction within some M_i or V which can again be handled by (C2) and the induction hypothesis

There are no other reductions and thus by (C3)
$\texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \{\ldots, (i,x).M_i, \ldots\} \in \mathrm{red}^c_{\underline{B}}$ □

The next step is to show that all terms are reducible if the free variables are substituted with reducible values.

Theorem 4.3.14 (Fundamental Property)**.**
If $\overrightarrow{x_i : A_i} \vdash^c M : B$ *and* $\forall i : V_i \in \mathrm{red}^v_{A_i}$ *then* $M[\overrightarrow{V_i/x_i}] \in red^c_{\underline{B}}$
and
if $\overrightarrow{x_i : A_i} \vdash^v V : A$ *and* $\forall i : V_i \in \mathrm{red}^v_{A_i}$ *then* $V[\overrightarrow{V_i/x_i}] \in red^v_A$

Proof. We prove both claims at once by induction on the typing derivation of an arbitrary term T

- If $T = x_i$ for some variable x_i, then $T[\overrightarrow{V_i/x_i}] = V_i$ is certainly reducible.
- If $T = M\ \texttt{to}\ x.N$ then $M[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{F\,A}$ and $N[\overrightarrow{V_i/x_i}, V/x] \in \mathrm{red}^c_{\underline{B}}$ for all $V \in \mathrm{red}^v_A$. By Lemma 4.3.6 $M[\overrightarrow{V_i/x_i}]\ \texttt{to}\ x.N[\overrightarrow{V_i/x_i}] = T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{\underline{B}}$
- If $T = M\ V$, then $M[\overrightarrow{V_i/x_i}] \in red^c_{A \to \underline{B}}$ and $V[\overrightarrow{V_i/x_i}] \in \mathrm{red}^v_A$ by induction hypothesis. By Lemma 4.3.9 $M[\overrightarrow{V_i/x_i}]\ V[\overrightarrow{V_i/x_i}] = T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_B$.
- If $T = \texttt{force}\ V$, then $V[\overrightarrow{V_i/x_i}] \in \mathrm{red}^v_{U\,\underline{B}}$ by induction hypothesis and $T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{\underline{B}}$ by Lemma 4.3.7.
- If $T = \texttt{pm}\ V\ \texttt{as}\ (x,y).M$, then $T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{\underline{B}}$ by induction hypothesis and Lemma 4.3.12.
- If $T = \texttt{pm}\ V\ \texttt{as}\ \{\ldots, (i,x).M_i, \ldots\}$, then $T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{\underline{B}}$ by induction hypothesis and Lemma 4.3.13.

- If $T = M\ \hat{\imath}$, then $T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{\underline{B}}$ by induction hypothesis and Lemma 4.3.11.
- If $T = \lambda x.N$, then $N[\overrightarrow{V_i/x_i}, V/x] \in \mathrm{red}^c_{\underline{B}}$ for all $V \in \mathrm{red}^v_A$ by induction hypothesis and thus $\lambda x.(N[\overrightarrow{V_i/x_i}]) = T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{A\to\underline{B}}$ by the definition of reducibility.
- If $T = \texttt{return}\ V$ then $V[\overrightarrow{V_i/x_i}] \in \mathrm{red}^v_A$ by induction hypothesis and $T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^c_{F\,A}$ by the definition of reducibility.
- If $T = \texttt{thunk}\ M$, then $M[\overrightarrow{V_i/x_i}] \in red^c_{\underline{B}}$ by induction hypothesis and $T[\overrightarrow{V_i/x_i}] \in \mathrm{red}^v_{U\,\underline{B}}$ by the definition of reducibility.
- The cases $T = (\hat{\imath}, V)$, $T = (V, V')$,and $T = \lambda\{\ldots, i.M_i, \ldots\}$ can all be handled like the case $T = \texttt{thunk}\ M$ or $T = \texttt{return}\ V$

□

Theorem 4.3.15 (Strong Normalization)**.** *All terms of the Call-By-Push-Value calculus are strongly normalizing.*

Proof. For closed terms the claim follows immediately from Theorem 4.3.14 and (C1). For any computation M with free variables $x_0, \ldots, x_k$ we can find a closed computation $N = \lambda x_0 \ldots \lambda x_k.M$ using lambda closure. This term is strongly normalizing by the reasoning above. Since $M \to M'$ implies $\lambda x_0 \ldots \lambda x_k.M \to \lambda x_0 \ldots \lambda x_k.M'$ we have $max(M) \leq max(N)$ and thus M is strongly normalizing. For any value V with free variables we build the lambda closure around the term $\texttt{return}\ V$ and reason as above. □

Chapter 5

Observations

In the last section we have proven strong normalization for Call-By-Push-Value. In this chapter we will look at the connection between the simply-typed λ-calculus and the CBPV calculus. We will introduce an embedding of the λ-calculus into CBPV, due to Levy, and analyze some of its properties. Furthermore we will investigate confluence of CBPV and show why it fails. We will use the following convention throughout the chapter:

Notation. CBN and CBV refer to the simply-typed λ-calculus with sums under their respective deterministic evaluation strategies evaluating to weak head normal form. $\lambda^{\to\times+}$ and $\lambda^{\to\times}$ refer to the simply-typed λ-calculus, with or without sum types, under full nondeterministic $\beta\eta$-reduction. The additional rules for $\lambda^{\to\times+}$ can be found in Table 5.1

5.1 Embedding the λ-calculus into CBPV

In our normalization proof we considered the CBPV calculus without complex values. The main motivation was that complex values are not needed to subsume CBN and CBV. Both languages can be suitably embedded into CBPV without complex values. Thus, we will also not use complex values to embed $\lambda^{\to\times+}$ or $\lambda^{\to\times}$ into CBPV with our nondeterministic reduction relation.

What we would like to obtain is an embedding $(-)^n$ from the simply-typed λ-calculus to CBPV that fulfills at least the following property:

Property 5.1.1. If $M \to M'$ in $\lambda^{\to\times+}$ then $M^n \to^+ (M')^n$ in CBPV

We certainly need the $\to^+$ in this property since CBPV is a lot more fine grain than $\lambda^{\to\times+}$, and thus will require additional reduction steps to remove, `force thunk` prefixes and so on. Since there are already two embeddings, one for CBN and one for CBV, we will evaluate their suitability as an embedding for $\lambda^{\to\times+}$ into CBPV

Terms and typing rules:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash (1, M) : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash (2, M) : A + B}$$

$$\frac{\Gamma \vdash M : A + B \qquad \Gamma, x : A \vdash N_1 : C \qquad \Gamma, x : B \vdash N_2 : C}{\Gamma \vdash \texttt{pm}\ M\ \texttt{as}\ \{(1, x).N_1, (2, x)N_2\} : C}$$

reductions:

$$\texttt{pm}\ (\hat{\imath}, M)\ \texttt{as}\ \{(1, x).N_1, (2, x).N_2\} \to N_{\hat{\imath}}[M/x]$$

$$\texttt{pm}\ M\ \texttt{as}\ \{(1, x).^x N[(1, x)/z], (2, x).^x N[(2, x)/z]\} \to N[M/z]$$

Table 5.1: Additional rules for sum types in $\lambda^{\to\times+}$

When considering CBN and CBV, the difference between these two evaluation strategies can be reduced to the question of what we may substitute for an identifier during reduction. In the case of CBV we only allow fully evaluated terms to be substituted for an identifier, where fully evaluated means evaluated to weak head normal form. This directly implies that the argument to a function must be evaluated before the corresponding β-reduction can happen, leading to a CBV strategy. In CBN we only allow fully unevaluated terms to be substituted for an identifier. This in turn means that the β-reduction for function application must occur before any reduction in the argument occurs, leading to a CBN strategy. Using the original deterministic semantics of CBPV, these evaluation orders can be enforced by suitable embeddings, which can be found in [Lev99].

Apart from enforcing the operational semantics of CBV and CBN, these embeddings can also be used to translate denotational and other semantics even in the presence of side effects. This leads to problems when trying to use these embeddings to embed the effect free $\lambda^{\to\times+}$-calculus into CBPV with our nondeterministic reduction relation. The reason for this is that in the presence of effects, some of the η laws of $\lambda^{\to\times+}$ are no longer valid. For example, the η law for sum types

$$N[M/z] = \texttt{pm}\ M\ \texttt{as}\ \{(1, x).^x N[(1, x)/z], (2, x).^x N[(2, x)/z]\}$$

is not valid in CBN with effects. To illustrate this, we add the effect of printing, such that a term now evaluates to a tuple of an output string and the resulting term. So consider the term

$$\texttt{pm}\ (\texttt{print}\ \text{“x”}; (\hat{\imath}, ()))\ \texttt{as}\ \{(1, x).(), (2, x).()\}$$

thus setting $M = \texttt{print}$ "x"$;(\hat{\imath}, ())$ and $N = ()$. Evaluating this term in CBN yields ("x", ()), whereas the η-reduced term $()$ does not create output when it is evaluated. In CBV however, the η-law for sums holds, because we are only allowed to substitute completely evaluated terms for a variable. Thus, the above term is no instance of the η-law for CBV, since M is not terminal. The η-law for $\lambda^{\to\times+}$, allowing every term to be substituted for an identifier, does not hold in either setting. In contrast to the η-law for sums, the η-law for function types holds in CBN but not in CBV.

This however means that, although CBPV does fulfill both laws, neither the CBV nor the CBN embedding allows both η-rules to be derived via the embedding, and, considering the above, it seems unlikely to find an embedding that does. The reason for this are the fundamental differences between the two paradigms, leading to completely different embeddings, which are difficult to combine in a meaningful manner.

In CBV and in the presence of effects, we have to distinguish between unevaluated terms or returners, which can still cause effects, and values, which are fully evaluated and may thus be substituted for an identifier. Thus, a CBV term that is terminal has two different embeddings; one into computation types and one into value types. Here we make use of the fact that in CBPV only values can be bound to an identifier. This way we can translate the types of CBV into the value types of CBPV. In CBN however, we bind values to unevaluated terms, so all CBN types are translated to computation types and we bind identifiers to thunks of computations.

For the simply typed λ-calculus without sums, things are easier. The η-laws for function types and projection products are both valid in CBN and so we can adapt the embedding for CBN provided by Levy in [Lev99] to embed $\lambda^{\to\times}$ into CBPV. The terms and types of $\lambda^{\to\times}$ can be embedded as in Table 5.2. We use the unit type as the single base type. Other base types can be included in a similar manner.

The question is how to characterize this embedding. We will prove Property 5.1.1, resticted to $\lambda^{\to\times}$. To prove this and the following theorem, we need the following definition and lemma:

Definition 5.1.2. We define a resticted reduction in CBPV

$$M \rightsquigarrow N :\iff M \to N \text{ using either } \texttt{force thunk } M \to M$$
$$\text{or } \texttt{thunk force } V \to V$$

Lemma 5.1.3 (Substitution and $(-)^n$)**.**

$$M^n[\texttt{thunk } N^n/x] \rightsquigarrow^* (M[N/x])^n$$

Proof. By induction on the structure of the term M. The only case where a reduction occurs is the case $M = x$. Here we have

$$x^n[\texttt{thunk } N^n/x] = \texttt{force } x[\texttt{thunk } N^n/x] = \texttt{force thunk } N^n$$
$$\rightsquigarrow N^n = (x[N/x])^n$$

Types:

C	C^n(a computation type)
1	$F\,1$
$A \to B$	$(U\,A^n) \to B^n$
$A_1 \times A_2$	$\prod_{i\in\{1,2\}} A_i$

Terms:

$x_0 : A_0 \dots x_k : A_k \vdash M : C$	$x_0 : U\,A_0 \dots x_k : U\,A_k \vdash^c M^n : C^n$
x	$\texttt{force}\ x$
$()$	$\texttt{return}\ ()$
$\lambda x.M$	$\lambda x.M^n$
$M\ N$	$M^n(\texttt{thunk}\ N^n)$
(M, N)	$\lambda\{1.M^n, 2.N^n\}$
$\pi_{\hat{\imath}} M$	$M\ \hat{\imath}$

Table 5.2: Embedding the λ-calculus into CBPV

In all other cases we push the substitution to the subterms and apply the induction hypothesis, as in the case $M = (\hat{\imath}, P)$.

$$\begin{aligned}(\hat{\imath}, P)^n[\texttt{thunk}\ N^n/x] &= \texttt{return}\ (\hat{\imath}, \texttt{thunk}\ P^n)[\texttt{thunk}\ N^n/x] \\ &= \texttt{return}\ (\hat{\imath}, \texttt{thunk}\ P^n[\texttt{thunk}\ N^n/x]) \\ &\overset{\textit{I.H.}}{\rightsquigarrow}^* \texttt{return}\ (\hat{\imath}, \texttt{thunk}\ (P[N/x])^n) = (\hat{\imath}, P[N/x])^n\end{aligned}$$

□

Theorem 5.1.4. *If $M \to M'$ in $\lambda^{\to\times}$ then $M^n \to^+ (M')^n$ in CBPV.*

Proof. By induction over the structure of M. The case $M = x$ is trivial, the other cases are:

$M = \lambda x.N$ **:** If $N \neq {}^xP\ x$ then $\lambda x.N \to \lambda x.N'$ with $N \to N'$ only, and we apply the induction hypothesis for N, obtainting $N^n \to^+ (N')^n$. Thus $M^n = \lambda.N^n \to^+ \lambda.(N')^n = (M')^n$ as well. If $N = {}^xP\ x$ then $\lambda x.{}^xP\ x \to {}^xP$ and

$$\begin{aligned}M^n = (\lambda x.{}^xP\ x)^n &= \lambda x.({}^xP)^n\ (\texttt{thunk}\ \texttt{force}\ x) \\ &\rightsquigarrow \lambda x.{}^x(P^n)\ x \xrightarrow{\eta} ({}^xP)^n\end{aligned}$$

where $({}^xP)^n = {}^x(P^n)$ holds since $(-)^n$ does not introduce free variables.

$M = P\ Q$ **:** If $P \neq \lambda x.N$ then reduction can only occur in P or in Q and we apply the induction hypothesis as above. If $P = \lambda x.N$ then $(\lambda x.N)\ Q \to N[Q/x]$ and

$$((\lambda x.N)\ Q)^n = (\lambda x.N^n)\ (\texttt{thunk}\ Q^n) \xrightarrow{\beta} (N^n[\texttt{thunk}\ Q^n/x])$$

By Lemma 5.1.3, $(N^n[\texttt{thunk}\ Q^n/x]) \rightsquigarrow^* (N[Q/x])^n$ as required.

$M = (P, Q)$ **:** If $(P, Q) \neq (\pi_1 N, \pi_2 N)$ then $M \to M'$ iff $P \to P'$ or $Q \to Q'$. Assume $P \to P'$ then

$$(P, Q)^n = \lambda\{1.P^n, 2.Q^n\} \overset{I.H.}{\to^+} \lambda\{1.(P')^n, 2.Q^n\} = (P', Q)^n$$

The other case is symmetric. If $(P, Q) = (\pi_1 N, \pi_2 N)$, then $(\pi_1 N, \pi_2 N) \xrightarrow{\eta} N$ and

$$(\pi_1 N, \pi_2 N)^n = \lambda\{1.N^n\ 1, 2.N^n\ 2\} \xrightarrow{\eta} N^n$$

as required.

$M = \pi_1 N$ **:** If $N \neq (P, Q)$ then reduction is confined to N and we can apply the induction hypothesis. If $N = (P, Q)$, then $\pi_1(P, Q) \to P$ and

$$(\pi_1(P, Q))^n = \lambda\{1.P^n, 2.Q^n\}\ 1 \xrightarrow{\beta} P^n$$

□

Using the normalization result on CBPV we can immediately conclude:

Corollary 5.1.5. *$\lambda^{\to\times}$ is strongly normalizing.*

The other direction of the theorem above does not hold. In the reduction of a term M^n we can even selectively keep redexes of the form `force thunk` N to "skip over" the images of all M' reachable from M. But if we investigate the above proof closely, we can use it to prove a stronger property. For this we define an equivalence relation on the terms of CBPV "modulo `force thunk`" and a corresponding reduction.

Definition 5.1.6 (Equivalence relation)**.**

- $M \sim N :\iff \exists P_1 \dots P_k : M \rightsquigarrow^* P_1 \reflectbox{$\rightsquigarrow$}^* \cdots \rightsquigarrow^* P_k N$
- $[M]_\sim \to [N]_\sim :\iff \exists P, Q : M \sim P \to Q \sim N$ where $\to$ is not a $\rightsquigarrow$-reduction.

Thus we can prove:

Theorem 5.1.7. *If $M \to M'$ in $\lambda^{\to\times}$ then $[M^n]_\sim \to [(M')^n]_\sim$ in CBPV.*

Proof. Any element of $[M^n]_\sim$ can be converted to M^n using only $\rightsquigarrow$-conversions. We then use the corresponding case of the proof of Theorem 5.1.4 where we use exactly one reduction that is not from $\rightsquigarrow$ □

From Theorem 5.1.4 we can also derive

Corollary 5.1.8. *If $M \to^* N$ in $\lambda^{\to\times}$ then $M^n \to^* N^n$*

Proof. The reflexive case is trivial, the case $M \to^k N$ follows by k-fold application of Theorem 5.1.4 □

For both Theorem 5.1.7 and Corollary 5.1.8 we conjecture the other direction to hold as well. This would allow confluence to be carried over from CBPV to $\lambda^{\to\times}$, provided that the normal form reached in CBPV is in $[M^n]_\sim$ for some M. While this may hold for the fragment of CBPV used by the embedding, we will see in the next section that, in general, CBPV is not confluent.

5.2 Failure of Confluence

The original definitions for reducibility in our normalization proof were made in the hope to generalize this technique to other observational predicates, like confluence. Unfortunately confluence for CBPV fails in the presence of the η-rules for the `pm`-constructs. We consider the following counterexample in detail, because it is a nice display of the power of the η-conversion rule. We define

$$\begin{aligned} T_1 = \texttt{pm}\ P_1\ \texttt{as}\ \{&(1,x).(\texttt{pm}\ P_2\ \texttt{as}\ \{(1,y).N_1, (2,y).N_2\}),\\ &(2,x).(\texttt{pm}\ P_2\ \texttt{as}\ \{(1,y).M_1, (2,y).M_2\})\} \end{aligned}$$

and

$$\begin{aligned} T_2 = \texttt{pm}\ P_2\ \texttt{as}\ \{&(1,y).(\texttt{pm}\ P_1\ \texttt{as}\ \{(1,x).N_1, (2,x).M_1\}),\\ &(2,y).(\texttt{pm}\ P_1\ \texttt{as}\ \{(1,x).N_2, (2,x).M_2\})\} \end{aligned}$$

where x and y do not occur free in P_i, M_i, N_i, dropping the usual annotation to improve readability. We can prove $T_1 = T_2$ within the CBPV equational theory using the rules of Table 2.3 in their undirected versions in the following way:
We write T_1 as $Q[P_2/z]$ giving us

$$\begin{aligned} Q = \texttt{pm}\ P_1\ \texttt{as}\ \{&(1,x).(\texttt{pm}\ z\ \texttt{as}\ \{(1,y).N_1, (2,y).N_2\}),\\ &(2,x).(\texttt{pm}\ z\ \texttt{as}\ \{(1,y).M_1, (2,y).M_2\})\} \end{aligned}$$

and then η-expand $Q[P_2/z]$ as

$$
\begin{aligned}
&\texttt{pm } P_2 \texttt{ as } \{\ldots,(i,k).{}^{k}Q[(i,k)/z],\ldots\} \\
=\ &\texttt{pm } P_2 \texttt{ as } \{\ (1,k).(\texttt{pm } P_1 \texttt{ as } \{(1,x).(\texttt{pm } (1,k) \texttt{ as } \{(1,y).N_1,(2,y).N_2\}),\\
&\qquad\qquad (2,x).(\texttt{pm } (1,k) \texttt{ as } \{(1,y).M_1,(2,y).M_2\})\}\\
&\qquad (2,k).(\texttt{pm } P_1 \texttt{ as } \{(1,x).(\texttt{pm } (2,k) \texttt{ as } \{(1,y).N_1,(2,y).N_2\}),\\
&\qquad\qquad (2,x).(\texttt{pm } (2,k) \texttt{ as } \{(1,y).M_1,(2,y).M_2\})\}
\end{aligned}
$$

for some fresh variable k. Using β-reduction we obtain

$$
\begin{aligned}
T_2 = \texttt{pm } P_2 \texttt{ as } \{&(1,k).(\texttt{pm } P_1 \texttt{ as } \{(1,x).N_1,(2,x).M_1\}),\\
&(2,k).(\texttt{pm } P_1 \texttt{ as } \{(1,x).N_2,(2,x).M_2\})\}
\end{aligned}
$$

which α-converts to T_2.

Thus, for confluence, we would require $T_1 \to^* T \leftarrow^* T_2$, for some term T. However, we can choose $P_i = x_i$, avoiding β-reduction, and chose all M_i and N_i to be distinct computations in normal form (such as $\texttt{return } x_j$ with distinct x_j for all M, N) avoiding η-reduction. Under these circumstances T_1 and T_2 are provably equal, syntactically different and both normal, contradicting confluence. Furthermore, T_1 and T_2 have the same syntactic structure. Thus, extending the reduction to include either $T_1 \to^* T_2$ or $T_2 \to^* T_1$ would make the calculus non-normalizing. An example similar to this one can also be derived for the pm rule for products.

This failure of confluence is not specific to CBPV. The same problem arises in the simply typed λ-calculus with sums in the presence of η-reduction. Lindley uses a decomposition of the η rule for sum types into several axioms to define a normalizing and confluent rewriting theory for $\lambda^{\to\times+}$ modulo some decidable congruence relation [Lin07]. This congruence relation contains the case above as a special case. As a corollary he obtains decidability of the $\lambda^{\to\times+}$ equational theory. A similar treatment of CBPV might also be possible.

5.3 Conclusions

We have introduced a new notion of reduction for the Call-By-Push-Value calculus and have investigated strong normalization and confluence of the resulting calculus.

For the proof of strong normalization we have introduced a new notion of $\top\top$-closure. While our $\top\top$-closure is not completely new and clearly related to the constructs of Lindley, Stark, and Pitts, we believe that our use in normalization proofs for calculi with multiple computation types is.

While we have proven strong normalization for CBPV without complex values, it is not clear how this proof can be adapted to the variant of CBPV allowing pattern-matching not only into computations but also into values. One of the problems arising in this context is how to suitably extend the

stacks to also accept values, while maintaining a well defined notion of reduction on stacks.

For confluence we have seen that the η-rules for sums and pattern-match products in combination with the usual β-rules make the calculus non-confluent. The problem is even more severe, as the addition of reduction rules establishing confluence would destroy the normalization result. Thus, the straightforward approach to proving decidability of the CBPV equational theory fails and we are not aware of any other work in that direction.

We have found an embedding of $\lambda^{\to\times}$ into CBPV that allows the well known normalization result of the λ-calculus to be derived from the corresponding result for CBPV. On the other hand, we also have seen that the handling of η-rules is not only problematic when considering confluence, they also cause problems when trying to embed $\lambda^{\to\times+}$ into the CBPV calculus. Further evaluation of this subject, possibly including CBPV with complex values, may also provide an embedding of $\lambda^{\to\times+}$ into CBPV.

Altogether, CBPV together with our nondeterministic reduction relation turns out be an interesting calculus whose properties are far from being fully understood.

Bibliography

[BBdP98] P.N. Benton, Gavin M. Bierman, and Valeria de Paiva. Computational types from a logical perspective. *Journal of Functional Programming*, 8(2):177–193, 1998.

[dG02] Philippe de Groote. On the strong normalisation of intuitionistic natural deduction with permutation-conversions. *Information and Computation*, 178:441 – 464, 2002.

[FF86] Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD - machine and the λ-calculus. In Martin Wirsing, editor, *Formal Description of Programming Concepts III*, Amsterdam - New York - Oxford, 1986. North-Holland.

[Gir72] Jean-Yves Girard. Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur. Thèse de doctorat d'état (Université de Paris VII), 1972.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Univ. Press, Cambridge, 1989.

[Jon03] Simon P. Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

[Lev99] Paul B. Levy. Call-by-push-value: A subsuming paradigm (extended abstact). In Jean-Yves Girard, editor, *Proceedings Typed Lambda Calculi and Applications (TLCA'99)*, volume 1581 of *Lecture Notes in Computer Science*, pages 228–242, 1999.

[Lev04] Paul B. Levy. *Call-By-Push-Value*, volume 2 of *Semantics Structures in Computation*. 2004.

[Lin05] Sam Lindley. *Normalisation by Evaluation in the Compilation of Typed Functional Programming Languages*. PhD thesis, University of Edinburgh, 2005.

[Lin07] Sam Lindley. Extensional rewriting with sums. In *Typed Lambda Calculi and Applications*, volume 4583 of *Lecture Notes in Computer Science*, pages 255–271. Springer-Verlag, 2007.

[LS05] Sam Lindley and Ian Stark. Reducibility and $\top\top$-lifting for computation types. In *Typed Lambda Calculi and Applications: Proceedings of the Seventh International Conference TLCA 2005*, volume 3461 of *Lecture Notes in Computer Science*, pages 262–277. Springer-Verlag, 2005.

[Mit96] John C. Mitchell. *Foundations for Programming Languages*. Foundations of computing series. MIT Press, Cambridge, Massachusetts - London, England, 1996.

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

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, Cambridge, Massachusetts - London, England, 2002.

[Pit00] Andrew M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.

[Pra71] Dag Prawitz. Ideas and results in proof theory. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundation of Mathematics*. North-Holland, 1971.

[PS98] Andrew M. Pitts and Ian Stark. Operational reasoning for functions with local state. In Andrew M. Pitts and Andrew D. Gordon, editors, *Higher order operational techniques in semantics*, pages 227–274. 1998.

[Tai67] William W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32, 1967.

[Tai72] William W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer-Verlag, Boston, 1972.