Call-By-Push-Value in Coq: Operational, Equational, and Denotational Theory

Yannick Forster
Saarland University
Saarbrücken, Germany
forster@ps.uni-saarland.de

Steven Schäfer
Saarland University
Saarbrücken, Germany
schaefer@ps.uni-saarland.de

Simon Spies
Saarland University
Saarbrücken, Germany
spies@ps.uni-saarland.de

Kathrin Stark
Saarland University
Saarbrücken, Germany
stark@ps.uni-saarland.de

Abstract

Call-by-push-value (CBPV) is an idealised calculus for functional and imperative programming, introduced as a subsuming paradigm for both call-by-value (CBV) and call-by-name (CBN). We formalise weak and strong operational semantics for (effect-free) CBPV, define its equational theory, and verify adequacy for the standard set/algebra denotational semantics. Furthermore, we prove normalisation of the standard reduction, confluence of strong reduction, strong normalisation using Kripke logical relations, and soundness of the equational theory using logical equivalence. We adapt and verify the known translations from CBV and CBN into CBPV for strong reduction. This yields, for instance, proofs of strong normalisation and confluence for the full λ-calculus with sums and products. Thanks to the automation provided by Coq and the Autosubst 2 framework, there is little formalisation overhead compared to detailed paper proofs.

CCS Concepts • **Theory of computation** → **Operational semantics**; **Denotational semantics**; **Type theory**.

Keywords Call-by-push-value, operational semantics, denotational semantics, equational theory, strong normalisation, formalisation, type theory, Coq

ACM Reference Format:
Yannick Forster, Steven Schäfer, Simon Spies, and Kathrin Stark. 2019. Call-By-Push-Value in Coq: Operational, Equational, and Denotational Theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '19), January 14–15, 2019, Cascais, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3293880.3294097

1 Introduction

Call-by-push-value (CBPV) is an idealised calculus for functional programming, suitable as a base calculus for the inclusion of effects and able to express both call-by-value (CBV) and call-by-name (CBN) reduction strategies [Levy 1999, 2012]. CBPV syntactically distinguishes between values and computations, following the mantra "A value is, a computation does". Due to its fixed evaluation order, there exist translations from CBV and CBN calculi into CBPV which preserve denotational and (big-step) operational semantics for simply-typed terms [Levy 2006]. Using the translations, results like weak normalisation for the simply-typed λ-calculus or adequacy of a denotational semantics can be obtained directly from the corresponding results for CBPV.

From a compiler perspective, CBV and CBN can be seen as source languages which are translated to the intermediate language CBPV. This is the approach recently proposed by the pioneering work of Rizkallah, Garbuzov, and Zdancewic [2018]. They are the first to formalise CBPV in Coq and formalise soundness of the equational theory (considering only β-laws) w.r.t. observational equivalence.

In this paper we aim to extend the previous results of both Levy [2006] and Rizkallah et al. [2018]. We consider translations from the call-by-name λ-calculus and the fine-grained call-by-value λ-calculus, untyped and with binary sums and products, into CBPV. The main contribution of this paper is two-fold:

First, we give a setup and examination of a strong operational semantics for CBPV, allowing reduction in every possible context. In the compiler setting of Rizkallah et al. [2018], strong reduction corresponds to partial evaluation and serves as the basis for many program optimisations, see e.g. [Leißa et al. 2015]. We show how CBPV corresponds to CBV/CBN via a slight modification of the known translations and transport several results, such as strong normalisation, to CBV/CBN. Moreover, we can directly conclude confluence of the full λ-calculus. We then show that the equational theory (with β, η, and sequencing laws) of simply-typed CBPV is sound with respect to observational equivalence. This also suffices to obtain soundness of the equational theories for CBN/CBV (with β laws). Second, our work is accompanied by a complete, technically involved Coq development with about 8000 lines. This yields a verified treatment of both weak and strong operational semantics, the equational theory, and denotational semantics for CBPV, CBV, and CBN.

Organisation of the paper. We start the paper with a presentation of the untyped and simply-typed systems. Levys' known translations from CBV/CBN to CBPV is slightly modified by an eager let operation that eliminates administrative let redices. The translation is then proven correct for untyped terms with respect to the standard small-step semantics. We then formalise weak normalisation of simply-typed CBPV using a logical relation and, using the translations, conclude weak normalisation of simply typed CBV and CBN.

Strong reduction for CBPV is defined by allowing reduction in any context. We prove confluence of this reduction using a standard method [Takahashi 1989].

The CBN system equipped with strong reduction is the full λ-calculus. Here, the simulation proof is even easier than before, yielding confluence of the full λ-calculus with sums and products. For CBV equipped with strong reduction we observe that the translation does not (and probably: cannot) preserve normality for untyped terms. However, forward simulation still yields the soundness of the equational theory for CBV and suffices to prove strong normalisation.

In CBPV, strong normalisation is proven using a new proof method, following the structure of *weak* normalisation: To obtain strong normalisation, we extend the logical relation to open terms and slightly adapt the logical relation for computations. The simulations from simply-typed CBV and CBN then yield proofs of strong normalisation for CBV and CBN.

Next, we recall and formalise observational equivalence for all three systems. To obtain soundness for the equational theory of CBPV including β-laws, η-laws and let-sequencing, we define a logical equivalence that can be placed between the equational theory and observational equivalence.

Observational equivalence for both CBV and CBN terms can be concluded from the observational equivalence of their translations.

Last, we recall and formalise the set/algebra semantics for CBPV following Forster et al. [2017] and prove its adequacy in Coq. Adequate algebra semantics can be deduced for both CBV and CBN.

We conclude the paper by a discussion of the Coq formalisation, related work, and future work.

Coq formalisation. The Coq formalisation of all results in this paper is available online[1]. All lemmas and theorems in the PDF version of the paper are hyperlinked with the Coq source code. For the paper presentation, we use named syntax for variables and binders and will sometimes write contexts as functions. In contrast, our formalisation uses well-scoped de Bruijn syntax and the Autosubst 2 tool [Stark et al. 2018]. The most notable point about the formalisation is its generally low overhead compared to paper proofs.

[1] https://www.ps.uni-saarland.de/extras/cbpv-in-coq/

$$\begin{array}{lrl}
\text{(value types)} & A, B & := 1 \mid A_1 \times A_2 \mid 0 \mid A_1 + A_2 \mid U\,C \\
\text{(computation types)} & C, D & := F\,A \mid A \to C \mid \top \mid C_1 \,\&\, C_2 \\
\text{(environments)} & \Gamma & := x_1 : A_1, \ldots, x_n : A_n
\end{array}$$

$$\begin{array}{lrl}
\text{(values)} & V, W & := x \mid () \mid (V_1, V_2) \mid \mathbf{inj}_i\, V \mid \{M\} \\
\text{(computations)} & M, N & := \mathbf{split}(V, x_1.x_2.M) \mid \mathbf{case}_0(V) \mid \langle\rangle \\
 & & \mid\ \mathbf{case}(V, x_1.M_1, x_2.M_2) \mid V! \\
 & & \mid\ \mathbf{return}\ V \mid \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \\
 & & \mid\ \lambda x.M \mid M\,V \\
 & & \mid\ \langle M_1, M_2 \rangle \mid \mathbf{prj}_i\, M
\end{array}$$

Figure 1. CBPV syntax

Contributions. We provide a Coq formalisation of much of the known operational meta-theory of CBPV, including the definition of small-step reduction, weak normalisation and adequacy of the set/algebra semantics. We give translations from untyped CBV/CBN to CBPV and verify them with respect to the standard small-step semantics. The translation corresponds to the original one, except for an eager let construct.

To the best of our knowledge, we are the first to give a confluent strong operational semantics for CBPV. We deduce strong normalisation of the full simply typed λ-calculus with sums and products by proving the translation to be correct for strong reduction.

A new and general proof method is introduced to prove strong normalisation for CBPV.

We obtain the soundness result for the equational theory from Rizkallah et al. [2018] for our typed version of CBPV and can deduce soundness of the equational theory for CBV and CBN.

Our development can be further seen as a large case study for the Autosubst 2 library, featuring three different calculi using mutual inductive definitions.

2 Translating CBV and CBN to CBPV

We consider CBPV, CBV, and CBN with binary products, binary sums, and unit. The given type systems are all simply-typed without type variables. In the scope of this work, we do not consider effects. Note that if effects are considered, products and sums of arbitrary finite arity can *not* be recovered from their binary counterparts. However, we are confident that our proofs can be generalised to arbitrary finite arities.

2.1 CBPV

Our results concern (simply-typed) CBPV as defined in figs. 1 and 2. Value types include the empty type, unit, binary sums, and binary products. Computation types include unit, binary products, and function types.

The natural way to turn a value $V : A$ into a computation returning this value is using the **return** V construct, yielding a term of type $F\,A$. The only way to examine the value of a value-returning computation is by using the **let** $x \leftarrow M$ **in** N

Value typing $\boxed{\Gamma \vdash V : A}$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash V_1 : A_1 \qquad \Gamma \vdash V_2 : A_2}{\Gamma \vdash (V_1, V_2) : A_1 \times A_2}$$

$$\frac{\Gamma \vdash V : A_i}{\Gamma \vdash \mathbf{inj}_i\ V : A_1 + A_2} \qquad \frac{\Gamma \vdash M : C}{\Gamma \vdash \{M\} : U\,C}$$

Computation typing $\boxed{\Gamma \vdash M : C}$

$$\frac{\Gamma \vdash V : A_1 \times A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash M : C}{\Gamma \vdash \mathbf{split}(V, x_1.x_2.M) : C} \qquad \frac{\Gamma \vdash V : 0}{\Gamma \vdash \mathbf{case}_0(V) : C}$$

$$\frac{\Gamma \vdash V : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash M_1 : C \qquad \Gamma, x_2 : A_2 \vdash M_2 : C}{\Gamma \vdash \mathbf{case}(V, x_1.M_1, x_2.M_2) : C}$$

$$\frac{\Gamma \vdash V : U\,C}{\Gamma \vdash V! : C} \qquad \frac{\Gamma \vdash V : A}{\Gamma \vdash \mathbf{return}\ V : F\,A}$$

$$\frac{\Gamma \vdash M : F\,A \qquad \Gamma, x : A \vdash N : C}{\Gamma \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : C} \qquad \frac{\Gamma, x : A \vdash M : C}{\Gamma \vdash \lambda x.M : A \to C}$$

$$\frac{\Gamma \vdash M : A \to C \qquad \Gamma \vdash V : A}{\Gamma \vdash M\,V : C} \qquad \frac{}{\Gamma \vdash \langle\rangle : \top}$$

$$\frac{\Gamma \vdash M_1 : C_1 \qquad \Gamma \vdash M_2 : C_2}{\Gamma \vdash \langle M_1, M_2\rangle : C_1 \,\&\, C_2} \qquad \frac{\Gamma \vdash M : C_1 \,\&\, C_2}{\Gamma \vdash \mathbf{prj}_i\ M : C_i}$$

Figure 2. CBPV typing

construct. Computations $M : C$ can be *thunked* and then treated as values using $\{M\} : U\,C$. The only way to resume a thunked computation is by *forcing* using $V!$.

Eager let. Extending Levy's results to a small-step semantics and untyped versions of the λ-calculus requires a small change in the translation. We introduce an eager let construct which immediately inserts a value if present and otherwise just behaves like **let**:

$$\begin{aligned}
\mathbf{let}\ x \Leftarrow \mathbf{return}\ V\ \mathbf{in}\ N \quad &:= N[V/x] \\
\mathbf{let}\ x \Leftarrow M\ \mathbf{in}\ N \quad &:= \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N
\end{aligned}$$

Eager lets are frequent when translating into practical intermediate languages. Similar eager elimination constructs could be introduced for all other value types, but we will only use eager elimination of type $F\,A$. For CBV, the eager let is crucial for the (untyped) correctness of the translation w.r.t. both strong and weak reduction.

Both typing and substitution are compatible with the eager let. The (derived) typing rule for eager let reads

$$\frac{\Gamma \vdash M : F\,A \qquad \Gamma, x : A \vdash N : C}{\Gamma \vdash \mathbf{let}\ x \Leftarrow M\ \mathbf{in}\ N : C}$$

Lemma 2.1. *The eager let is compatible with renaming and substitution. Explicitly, for every substitution σ,*

$$(\mathbf{let}\ x \Leftarrow V\ \mathbf{in}\ M)[\sigma] = \mathbf{let}\ x \Leftarrow V[\sigma]\ \mathbf{in}\ M[\sigma[x := x]].$$

$$\begin{array}{lll}
\text{(types)} & A, B & := 1 \mid A_1 \times A_2 \mid A_1 + A_2 \mid A_1 \to A_2 \\
\text{(environments)} & \Gamma & := x_1 : A_1, \ldots, x_n : A_n
\end{array}$$

$$\begin{array}{lll}
\text{(values)} & u, v & := x \mid () \mid (v_1, v_2) \mid \mathbf{inj}_i\ v \mid \lambda x.s \\
\text{(terms)} & s, t & := \mathbf{val}\ v \\
& & \mid\ \mathbf{case}(s, x_1.t_1, x_2.t_2) \\
& & \mid\ \mathbf{split}(s, x_1.x_2.s) \mid s\ t
\end{array}$$

Figure 3. CBV syntax

Value typing $\boxed{\Gamma \vdash_v v : A}$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash_v x : A} \qquad \frac{}{\Gamma \vdash_v () : 1} \qquad \frac{\Gamma, x : A \vdash_e s : B}{\Gamma \vdash_v \lambda x.s : A \to B}$$

$$\frac{\Gamma \vdash_v v_1 : A_1 \qquad \Gamma \vdash_v v_2 : A_2}{\Gamma \vdash_v (v_1, v_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash_v v : A_i}{\Gamma \vdash_v \mathbf{inj}_i\ v : A_1 + A_2}$$

Computation typing $\boxed{\Gamma \vdash_e s : A}$

$$\frac{\Gamma \vdash_v v : A}{\Gamma \vdash_e \mathbf{val}\ v : A} \qquad \frac{\Gamma \vdash_e s : A \to B \qquad \Gamma \vdash_e t : A}{\Gamma \vdash_e s\ t : B}$$

$$\frac{\Gamma \vdash_e s : A_1 \times A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash_e t : C}{\Gamma \vdash_e \mathbf{split}(s, x_1.x_2.t) : C}$$

$$\frac{\Gamma \vdash_e s : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash_e t_1 : B \qquad \Gamma, x_2 : A_2 \vdash_e t_2 : B}{\Gamma \vdash_e \mathbf{case}(s, x_1.t_1, x_2.t_2) : B}$$

Figure 4. CBV typing

CBV translation $\boxed{\overline{A} \text{ and } \overline{v} \text{ and } \overline{s}}$

$$\overline{1} := 1 \qquad \overline{A \to B} := U\,(\overline{A} \to F\,\overline{B}) \qquad \overline{A \times B} := \overline{A} \times \overline{B}$$

$$\overline{A + B} := \overline{A} + \overline{B} \qquad \overline{()} := () \qquad \overline{x} := x \qquad \overline{\lambda x.s} := \{\lambda x.\overline{s}\}$$

$$\overline{(u, v)} := (\overline{u}, \overline{v}) \qquad \overline{\mathbf{inj}_b\ v} := \mathbf{inj}_b\ \overline{v} \qquad \overline{\mathbf{val}\ v} := \mathbf{return}\ \overline{v}$$

$$\overline{s\ t} := \mathbf{let}\ x \Leftarrow \overline{s}\ \mathbf{in}\ \mathbf{let}\ y \Leftarrow \overline{t}\ \mathbf{in}\ (x!)\ y$$

$$\overline{\mathbf{split}(s, x_1.x_2.t)} := \mathbf{let}\ z \Leftarrow \overline{s}\ \mathbf{in}\ \mathbf{split}(z, x_1.x_2.\overline{t})$$

$$\overline{\mathbf{case}(s, x_1.t_1, x_2.t_2)} := \mathbf{let}\ z \Leftarrow \overline{s}\ \mathbf{in}\ \mathbf{case}(z, x_1.\overline{t_1}, x_2.\overline{t_2})$$

Figure 5. CBV translation to CBPV

2.2 CBV

Levy [2006] gives a translation from the simply-typed fine-grained call-by-value λ-calculus (CBV, see figs. 3 and 4) to simply-typed CBPV and proves its correctness w.r.t. big-step operational semantics. We extend this result in that we prove that the translation is correct w.r.t. small-step semantics, and using eager lets to eliminate administrative redices. Levy [2006] presents his result only for typed terms, but remarks that they also hold for untyped terms, which we demonstrate with our formalisation.

$$
\begin{array}{lll}
\text{(types)} & A, B & := 1 \mid A_1 \times A_2 \mid A_1 + A_2 \mid A_1 \to A_2 \\
\text{(environments)} & \Gamma & := x_1 : A_1, \dots, x_n : A_n
\end{array}
$$

$$
\begin{array}{lll}
\text{(terms)} & s, t & := x \mid () \mid (s_1, s_2) \mid \mathbf{inj}_i\ s \\
& & \mid\ \mathbf{case}(s, x_1.t_1, x_2.t_2) \\
& & \mid\ \mathbf{prj}_i\ s \\
& & \mid\ \lambda x.s \mid s\ t
\end{array}
$$

Figure 6. CBN syntax

CBN typing $\boxed{\Gamma \vdash_n s : A}$

$$
\frac{(x : A) \in \Gamma}{\Gamma \vdash_n x : A}
\qquad
\frac{}{\Gamma \vdash_n () : 1}
\qquad
\frac{\Gamma, x : A \vdash_n s : B}{\Gamma \vdash_n \lambda x.s : A \to B}
$$

$$
\frac{\Gamma \vdash_n s_1 : A_1 \qquad \Gamma \vdash_n s_2 : A_2}{\Gamma \vdash_n (s_1, s_2) : A_1 \times A_2}
\qquad
\frac{\Gamma \vdash_n s : A_i}{\Gamma \vdash_n \mathbf{inj}_i\ s : A_1 + A_2}
$$

$$
\frac{\Gamma \vdash_n s : A \to B \qquad \Gamma \vdash_n t : A}{\Gamma \vdash_n s\ t : B}
\qquad
\frac{\Gamma \vdash_n s : A_1 \times A_2}{\Gamma \vdash_n \mathbf{prj}_i\ s : A_i}
$$

$$
\frac{\Gamma \vdash_n s : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash_n t_1 : C \qquad \Gamma, x_2 : A_2 \vdash_n t_2 : C}{\Gamma \vdash_n \mathbf{case}(s, x_1.t_1, x_2.t_2) : C}
$$

Figure 7. CBN typing

The types of CBV are translated into CBPV value types (fig. 5). The separation into terms and values in CBV is preserved in the translation: CBV terms and CBV values are translated to CBPV computations and CBPV values respectively. We write the translation from CBV types and CBV terms as $\overline{A}$, respectively $\overline{s}$ and $\overline{v}$. As the environments of both CBV and CBPV contain values, a translated environment simply translates the contained values: $\overline{\Gamma}x := \overline{\Gamma x}$.

We can show basic properties of our translation.

Lemma 2.2. *The translations are injective.*

Proof. By mutual induction on the corresponding first term or value. The proof relies on the fact that the eager let only substitutes variables which occur exactly once. □

Lemma 2.3. *If $\Gamma \vdash_e s : A$, then $\overline{\Gamma} \vdash \overline{s} : \overline{A}$ and analogously for values.*

Lemma 2.4. *The translations are compatible with substitution. Explicitly, for every substitution σ we have $\overline{s}[\overline{\sigma}] = \overline{s[\sigma]}$ and $\overline{v}[\overline{\sigma}] = \overline{v[\sigma]}$.*

2.3 CBN

Analogous to CBV, we give a translation from the call-by-name λ-calculus (CBN, see figs. 6 and 7) to simply-typed CBPV. Levy [2006] proves the correctness of the translation w.r.t. big-step operational semantics, while we extend the result as in section 2.2.

We write the translation functions as $\overline{s}$ and $\overline{A}$, their definitions are depicted in fig. 8. Note that all terms are translated to computations and all types to computation types. We translate environments Γ into value environments $\overline{\Gamma}x := U\,(\overline{\Gamma x})$.

CBN translation $\boxed{\overline{A} \text{ and } \overline{s}}$

$$
\overline{1} := F\,1 \qquad \overline{A_1 \times A_2} := \overline{A_1}\ \&\ \overline{A_2} \qquad \overline{A_1 + A_2} := F\,(U\,\overline{A_1} + U\,\overline{A_2})
$$

$$
\overline{A_1 \to A_2} := U\,\overline{A_1} \to \overline{A_2}
$$

$$
\overline{x} := x! \qquad \overline{()} := \mathbf{return}\ () \qquad \overline{(s_1, s_2)} := \langle \overline{s_1}, \overline{s_2} \rangle
$$

$$
\overline{\mathbf{inj}_i\ s} := \mathbf{return}\ \mathbf{inj}_i\ \{\overline{s}\}
$$

$$
\overline{\mathbf{case}(s, x_1.t_1, x_2.t_2)} := \mathbf{let}\ y \Leftarrow \overline{s}\ \mathbf{in}\ \mathbf{case}(y, x_1.\overline{t_1}, x_2.\overline{t_2})
$$

$$
\overline{\mathbf{prj}_i\ s} := \mathbf{prj}_i\ \overline{s} \qquad \overline{\lambda x.s} := \lambda x.\overline{s} \qquad \overline{s\ t} := \overline{s}\ \{\overline{t}\}
$$

Figure 8. CBN translation to CBPV

Following Levy we define the relation $s \mapsto_n M$ with the same defining rules as the translation function. We give the full definition in the appendix (fig. 3 in [Forster et al. 2018]) and only show the additional rule which allows force-thunk pairs at arbitrary positions:

$$
\frac{s \mapsto_n M}{s \mapsto_n \{M\}!}
$$

We fix some properties of the translation:

Lemma 2.5. *$\mapsto_n$ is injective.*

Lemma 2.6. *$s \mapsto_n \overline{s}$*

Lemma 2.7. *If $s \mapsto_n M$ and $\Gamma \vdash s : A$, then $\overline{\Gamma} \vdash M : \overline{A}$.*

Proof. By induction on $s \mapsto_n M$ with A generalised. □

Lemma 2.8. *Let ρ be a renaming. Then (1) $(\overline{s})[\rho] = \overline{s[\rho]}$ and (2) if $s \mapsto_n M$, then $s[\rho] \mapsto_n M[\rho]$.*

Lemma 2.9. *For all CBN substitutions σ and CBPV substitutions τ: Let $s \mapsto_n M$. If for all x, $\sigma x \mapsto_n \tau x!$, then $s[\sigma] \mapsto_n M[\tau]$.*

Finally, we introduce the full simulation relation $\Mapsto_n$. It has exactly the same rules as $\mapsto_n$, with the rule for **case** replaced by two new rules:

$$
\frac{s \Mapsto_n M \qquad t_1 \Mapsto_n N_1 \qquad t_2 \Mapsto_n N_2}{\mathbf{case}(s, x_1.t_1, x_2.t_2) \Mapsto_n \mathbf{let}\ y \leftarrow M\ \mathbf{in}\ \mathbf{case}(y, x_1.N_1, x_2.N_2)}
$$

$$
\frac{s \Mapsto_n \mathbf{return}\ V \qquad t_1 \Mapsto_n N_1 \qquad t_2 \Mapsto_n N_2}{\mathbf{case}(s, x_1.t_1, x_2.t_2) \Mapsto_n \mathbf{case}(M, x_1.N_1, x_2.N_2)}
$$

Lemma 2.10. *If $s \mapsto_n M$, then $s \Mapsto_n M$.*

Lemma 2.11. *If $s \Mapsto_n M$, then $s[\rho] \Mapsto_n M[\rho]$ for all renamings ρ.*

Lemma 2.12. *For all CBN substitutions σ and CBPV substitutions τ: Let $s \Mapsto_n M$. If for all x, $\sigma x \Mapsto_n \tau x!$, then $s[\sigma] \Mapsto_n M[\tau]$.*

Primitive reduction $\boxed{M \succ M'}$

$$\begin{aligned}
\textbf{split}((V_1, V_2), x_1.x_2.M) &\succ M[V_1/x_1, V_2/x_2]\\
\textbf{case}(\textbf{inj}_i\ V, x_1.M_1, x_2.M_2) &\succ M_i[V/x_i]\\
\{M\}! &\succ M\\
\textbf{let } x \leftarrow \textbf{return } V \textbf{ in } M &\succ M[V/x]\\
(\lambda x.M)\ V &\succ M[V/x]\\
\textbf{prj}_i\ \langle M_1, M_2\rangle &\succ M_i
\end{aligned}$$

Weak CBPV reduction frames

$$C := \textbf{let } x \leftarrow [\,] \textbf{ in } N \mid [\,]\ V \mid \textbf{prj}_i\ [\,]$$

Weak CBPV reduction $\boxed{M \rightsquigarrow M'}$

$$\frac{M \succ M'}{M \rightsquigarrow M'} \qquad \frac{M \rightsquigarrow M'}{C[M] \rightsquigarrow C[M']}$$

Figure 9. Weak CBPV reduction

3 Weak Reduction

We write the standard small-step semantics of CBPV (see e.g. [Forster et al. 2017]) as $\rightsquigarrow$. The corresponding semantics for CBV and CBN are denoted by $\rightsquigarrow_v$ and $\rightsquigarrow_n$.

We say that a term is *normal* if it does not have a successor under $\rightsquigarrow$, and say that M *evaluates to* N, written $M \Downarrow N$, iff $M \rightsquigarrow^* N \land \text{normal } N$ (and similarly for $\Downarrow_v$ / $\Downarrow_n$). In the Coq development we prove that this is equivalent to the usual big-step definition of evaluation for typed terms, but do not mention big-step evaluation here again. We use the same notation for evaluation in CBV and CBN.

3.1 CBPV

We define $\rightsquigarrow$ in fig. 9 and fix the following properties of eager let for future use:

Lemma 3.1. $(\textbf{let } x \leftarrow M \textbf{ in } N) \rightsquigarrow^* (\textbf{let } x \Leftarrow M \textbf{ in } N)$.

Lemma 3.2. *If* $M \rightsquigarrow^* M'$*, then* $\textbf{let } x \Leftarrow M \textbf{ in } N \rightsquigarrow^* \textbf{let } x \Leftarrow M' \textbf{ in } N$ *and similarly for* $\rightsquigarrow^+$.

3.2 CBV

See fig. 10 for a small-step semantics for CBV. We show that the translation of CBV into CBPV is a simulation, i.e. that $s \Downarrow_v v \leftrightarrow \overline{s} \Downarrow \overline{v}$.

The forward direction of the simulation is by two straightforward inductions:

Lemma 3.3. *We have:*

1. *If* $s \rightsquigarrow_v t$*, then* $\overline{s} \rightsquigarrow^+ \overline{t}$.
2. *If* $s \rightsquigarrow_v^* t$*, then* $\overline{s} \rightsquigarrow^* \overline{t}$.

Theorem 3.4. *If* normal s*, then* normal $\overline{s}$.

Corollary 3.5. *If* $s \Downarrow_v t$*, then* $\overline{s} \Downarrow \overline{t}$.

For the backwards direction, termination follows directly from forward simulation (Lemma 3.3).

Primitive CBV reduction $\boxed{s \succ_v s'}$

$$\begin{aligned}
\textbf{split}((v_1, v_2), x_1.x_2.M) &\succ_v M[v_1/x_1, v_2/x_2]\\
\textbf{case}(\textbf{inj}_i\ v, x_1.s_1, x_2.s_2) &\succ_v s_i[v/x_i]\\
(\lambda x.s)\ (\textbf{val}\ v) &\succ_v s[v/x]
\end{aligned}$$

Weak CBV reduction frames

$$C := (\textbf{val}\ v)\ [\,] \mid [\,]\ s \mid \textbf{case}([\,], x_1.t_1, x_2.t_2) \mid \textbf{split}([\,], x_1.x_2.t)$$

Weak CBV reduction $\boxed{s \rightsquigarrow_v s'}$

$$\frac{s \succ s'}{s \rightsquigarrow_v s'} \qquad \frac{s \rightsquigarrow_v s'}{C[s] \rightsquigarrow_v C[s']}$$

Figure 10. Weak CBV reduction

Primitive CBN reduction $\boxed{M \succ_n M'}$

$$\begin{aligned}
\textbf{case}(\textbf{inj}_i\ s, x_1.t_1, x_2.t_2) &\succ_n t_i[s/x_i]\\
(\lambda x.s)\ t &\succ_n s[t/x]\\
\textbf{prj}_i\ (s_1, s_2) &\succ_n s_i
\end{aligned}$$

Weak CBN reduction frames

$$C := [\ \,]\ s \mid \textbf{prj}_i\ [\ \,] \mid \textbf{case}([\ \,], x_1.t_1, x_2.t_2)$$

Weak CBN reduction $\boxed{M \rightsquigarrow_n M'}$

$$\frac{M \succ_n M'}{M \rightsquigarrow_n M'} \qquad \frac{M \rightsquigarrow_n M'}{C[M] \rightsquigarrow_n C[M']}$$

Figure 11. Weak CBN reduction

Corollary 3.6. *If* normal $\overline{s}$*, then* normal s.

For backwards simulation, we need to show that if $\overline{s} \rightsquigarrow^* \overline{t}$, then $s \rightsquigarrow^* t$.

Lemma 3.7. *If* $\overline{s} \rightsquigarrow M$*, then there is some* s' *such that* $M \rightsquigarrow^* \overline{s'}$ *and* $s \rightsquigarrow_v s'$.

Proof. By induction on s. □

Lemma 3.8. *If* $\overline{s} \rightsquigarrow^* M$*, then there is some* t *such that* $M \rightsquigarrow^* \overline{t}$ *and* $s \rightsquigarrow_v^* t$.

Proof. With functionality of weak CBPV reduction in the induction case. □

The full backward simulation result requires injectivity of the translation.

Corollary 3.9. *If* $\overline{s} \Downarrow \overline{v}$*, then* $s \Downarrow_v v$.

3.3 CBN

For CBN, we can only show that the simulation holds up to the translation relation: We prove that $s \rightsquigarrow_n t$ implies

$\overline{s} \rightsquigarrow N$ for some N with $t \mapsto_n N$, but not necessarily $N = \overline{t}$. To see the difference to CBV, consider $s = (\lambda xy.x)u$ for a closed term u. We have $s \rightsquigarrow_n \lambda y.\overline{u}$. But at the same time, we have $\overline{s} = (\lambda xy.x!)\{\overline{u}\} \rightsquigarrow_n \lambda y.\{\overline{u}\}! \neq \lambda y.\overline{u}$.

The forward direction of the simulation still holds by a straightforward induction on the translation relation:

Lemma 3.10 (Forward simulation)**.**

1. *If $s \mapsto_n M$ and $t \mapsto_n N$, then $s[t/x] \mapsto_n M[\{N\}/x]$.*
2. *If $s \rightsquigarrow_n t$ and $s \mapsto_n M$, then $M \rightsquigarrow^+ N$ for some N with $t \mapsto_n N$.*
3. *If $s \rightsquigarrow_n^* t$ and $s \mapsto_n M$, then $M \rightsquigarrow^* N$ for some N with $t \mapsto_n N$.*

We can use the simulation to recover Levy's results about evaluation:

Lemma 3.11. *If $s \mapsto_n M$ and s is normal, then $M \rightsquigarrow^* N$ for a normal N with $s \mapsto_n N$.*

Theorem 3.12. *If $s \mapsto_n M$ and $s \Downarrow_n t$, then $M \Downarrow N$ for some N with $t \mapsto_n N$.*

For the backwards direction we would like to prove that if $\overline{s} \rightsquigarrow^* \overline{t}$, then $s \rightsquigarrow_n^* t$. For the induction, a generalisation to $\mapsto_n$ does not suffice: Consider

$$s = (\lambda x.\mathbf{case}(x, y_1.t_1, y_2.t_2))(u_1, u_2).$$

Then $\overline{s} \rightsquigarrow^* \mathbf{case}(\mathbf{return}\ (u_1, u_2), y_1.t_1, y_2.t_2)$, which is never contained in the translation relation. We thus generalise to the full simulation relation $\Mapsto_n$, which includes these intermediate terms:

Lemma 3.13. *The following hold:*

1. *If $s \Mapsto_n M$ and $M \rightsquigarrow N$, then there is t with $t \Mapsto_n N$ and $s \rightsquigarrow_n^* t$.*
2. *If $s \Mapsto_n M$ and $M \rightsquigarrow^* N$, then there is t with $t \Mapsto_n N$ and $s \rightsquigarrow_n^* t$.*

This is enough to deduce the backwards direction w.r.t. $\mapsto_n$:

Corollary 3.14. *If $s \mapsto_n M$, $t \mapsto_n N$ and $M \rightsquigarrow^* N$, then $s \rightsquigarrow_n^* t$.*

Finally, we can recover the results for evaluation:

Lemma 3.15. *If $s \Mapsto_n M$ and $s \rightsquigarrow_n t$, then there is t with $t \Mapsto_n N$ and $M \rightsquigarrow^+ N$.*

Theorem 3.16. *If $s \Mapsto_n M$ and $M \Downarrow N$, then there is t with $t \Mapsto_n N$ and $s \Downarrow_n t$.*

4 Weak Normalisation

Recall that for all three calculi weak reduction $\rightsquigarrow$ is deterministic. This allows us to define weak normalisation as strong normalisation of $\rightsquigarrow$. This definition is particularly nice in Coq, because an inductive definition of strong normalisation enables compact and elegant proofs by providing convenient induction hypotheses. We say that R is strongly normalising if SN $R\ x$ holds for all x, where SN is defined as:

$$\frac{\forall y.\ R\,x\,y \to \mathrm{SN}\ R\ y}{\mathrm{SN}\ R\ x}$$

A computation M is *weakly normalising*, if SN $(\rightsquigarrow)$ M, and similarly for CBN and CBV.

Value Semantic Typing $\boxed{V \in \mathcal{V}[A]}$

$$\mathcal{V}[0] := \emptyset \qquad \mathcal{V}[1] := \{()\}$$

$$\mathcal{V}[A_1 \times A_2] := \{(V_1, V_2) \mid V_1 \in \mathcal{V}[A_1], V_2 \in \mathcal{V}[A_2]\}$$

$$\mathcal{V}[A_1 + A_2] := \{\mathbf{inj}_i\ V \mid V \in \mathcal{V}[A_i]\}$$

$$\mathcal{V}[U\,C] := \{\{M\} \mid M \in \mathcal{E}[C]\}$$

Computation Semantic Typing $\boxed{M \in \mathcal{C}[C]}$

$$\mathcal{C}[\top] := \{\langle\rangle\} \qquad \mathcal{C}[F\,A] := \{\mathbf{return}\ V \mid V \in \mathcal{V}[A]\}$$

$$\mathcal{C}[A \to C] := \{\lambda x.M \mid \forall V \in \mathcal{V}[A].\ M[V/x] \in \mathcal{E}[C]\}$$

$$\mathcal{C}[C_1 \,\&\, C_2] := \{\langle M_1, M_2\rangle \mid M_1 \in \mathcal{E}[C_1], M_2 \in \mathcal{E}[C_2]\}$$

Semantic Typing

$$\mathcal{E}[C] := \{M \mid \exists N.\ M \Downarrow N \land N \in \mathcal{C}[C]\}$$

$$\mathcal{G}[\Gamma] := \{\gamma \mid \forall (x : A) \in \Gamma, \gamma x \in \mathcal{V}[A]\}$$

$$\Gamma \vDash V : A := \forall \gamma \in \mathcal{G}[\Gamma].\ V[\gamma] \in \mathcal{V}[A]$$

$$\Gamma \vDash M : C := \forall \gamma \in \mathcal{G}[\Gamma].\ M[\gamma] \in \mathcal{E}[C]$$

Figure 12. CBPV semantic typing

We prove weak normalisation of simply typed CBPV using logical relations [Girard 1971; Tait 1967]. We use the unary semantic typing predicates $\mathcal{V}[A]$ and $\mathcal{C}[C]$ for values and computation and introduce the unary expression relation $\mathcal{E}[C]$ following Dreyer et al. [2018] (all defined in fig. 12). The terms in $\mathcal{E}[C]$ are computationally indistinguishable from closed terms of type C. However, they are not necessarily well-typed. For example, $\mathbf{prj}_1\ \langle\langle\rangle, \langle\rangle\ ()\rangle \in \mathcal{E}[\top]$. Terms in $\mathcal{V}[A]$ and $\mathcal{C}[C]$ behave like closed normal forms of type A and C respectively.

Lemma 4.1. *The following hold:*

1. *$\mathcal{C}[C] \subseteq \mathcal{E}[C]$.*
2. *$\mathcal{C}[C]$ only contains normal forms w.r.t. $\rightsquigarrow$.*
3. *If $M \rightsquigarrow^* N$ and $N \in \mathcal{E}[C]$, then $M \in \mathcal{E}[C]$.*

Theorem 4.2. *If $\Gamma \vdash M : C$, then $\Gamma \vDash M : C$ and if $\Gamma \vdash V : A$, then $\Gamma \vDash V : A$.*

Proof. By induction on the typing judgement using compatibility lemmas for every case. ◻

Corollary 4.3. *If $\vdash M : C$, then $M \in \mathcal{E}[C]$ and if $\vdash V : A$, then $V \in \mathcal{V}[A]$.*

This suffices to deduce:

Strong CBPV reduction frames

$$
\begin{array}{lll}
\text{(value/value contexts)} & C_{v,v} & ::= [\,] \mid ([\,], V_2) \mid (V_1, [\,]) \mid \mathbf{inj}_i\,[\,] \\
\text{(value/computation contexts)} & C_{v,c} & ::= \{[\,]\} \\
\text{(computation/computation contexts)} & & \\
 & C_{c,c} & ::= [\,] \mid \mathbf{split}(V, x_1.x_2.[\,]) \\
 & & \mid\ \mathbf{case}(V, x_1.[\,], x_2.M_2) \\
 & & \mid\ \mathbf{case}(V, x_1.M_1, x_2.[\,]) \\
 & & \mid\ \mathbf{let}\ x \leftarrow [\,]\ \mathbf{in}\ N \\
 & & \mid\ \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ [\,] \\
 & & \mid\ \lambda x.[\,] \mid [\,]\,V \\
 & & \mid\ \langle [\,], M_2\rangle \mid \langle M_1, [\,]\rangle \mid \mathbf{prj}_i\,[\,] \\
\text{(computation/value contexts)} & & \\
 & C_{c,v} & ::= \mathbf{split}([\,], x_1.x_2.M) \\
 & & \mid\ \mathbf{case}_0([\,]) \\
 & & \mid\ \mathbf{case}([\,], x_1.M_1, x_2.M_2) \mid [\,]! \\
 & & \mid\ \mathbf{return}\ [\,] \mid M\,[\,]
\end{array}
$$

Strong CBPV reduction $\boxed{M \rightsquigarrow M' \ \text{ and } \ V \rightsquigarrow V'}$

$$
\frac{M \succ M'}{M \rightsquigarrow M'} \qquad \frac{M \rightsquigarrow M'}{C_{c,c}[M] \rightsquigarrow C_{c,c}[M']}
$$

$$
\frac{V \rightsquigarrow V'}{C_{c,v}[V] \rightsquigarrow C_{c,v}[V']} \qquad \frac{M \rightsquigarrow M'}{C_{v,c}[M] \rightsquigarrow C_{v,c}[M']}
$$

$$
\frac{V \rightsquigarrow V'}{C_{v,v}[V] \rightsquigarrow C_{v,v}[V']}
$$

Figure 13. Strong CBPV reduction

Theorem 4.4. *Closed, well-typed CBPV computations are weakly normalising.*

Weak normalisation for simply-typed CBV and CBN follow with the simulation theorems 3.3 and 3.10.

Theorem 4.5. *Closed, well-typed CBV terms are weakly normalising.*

Theorem 4.6. *Closed, well-typed CBN terms are weakly normalising.*

5 Strong Reduction

We define strong reduction for CBPV ($\rightsquigarrow$), CBV ($\rightsquigarrow_{\mathrm{v}}$), and CBN ($\rightsquigarrow_n$). We show confluence for CBPV using standard methods [Takahashi 1989] and adapt this result for the full λ-calculus via untyped simulation. For CBV the translation is not a simulation w.r.t. strong reduction, and thus we cannot deduce confluence for CBV from that of CBPV. However, forward simulation will suffice for strong normalisation and soundness of the equational theory.

5.1 CBPV

We extend weak reduction to strong reduction by allowing primitive reduction in every context, see fig. 13. We define strong reduction contexts C non-recursively, as single-layered frames. For real contexts, i.e. multi-layered frames we write $\mathcal{X}_{c,c}, \mathcal{X}_{c,v}, \mathcal{X}_{v,c}, \mathcal{X}_{v,v}$.

Strong reduction frames

$$
\begin{array}{lll}
\text{(value/value contexts)} & C_{v,v} & ::= [\,] \mid (v_1, [\,]) \mid ([\,], v_2) \\
 & & \mid\ \mathbf{inj}_i\,[\,] \\
\text{(value/computation contexts)} & C_{v,c} & ::= \lambda x.[\,] \\
\text{(computation/computation contexts)} & & \\
 & C_{c,c} & ::= [\,] \mid s\,[\,] \mid [\,]\,t \\
 & & \mid\ \mathbf{case}([\,], x_1.t_1, x_2.t_2) \\
 & & \mid\ \mathbf{case}(s, x_1.[\,], x_2.t_2) \\
 & & \mid\ \mathbf{case}(s, x_1.t_1, x_2.[\,]) \\
 & & \mid\ \mathbf{split}([\,], x_1.x_2.t) \\
 & & \mid\ \mathbf{split}(s, x_1.x_2.[\,]) \\
\text{(computation/value contexts)} & & \\
 & C_{c,v} & ::= \mathbf{val}\,[\,]
\end{array}
$$

Strong CBV reduction $\boxed{s \rightsquigarrow_{\mathrm{v}} s' \ \text{and}\ v \rightsquigarrow_{\mathrm{v}} v'}$

$$
\frac{s \succ_{\mathrm{v}} s'}{s \rightsquigarrow_{\mathrm{v}} s'} \qquad \frac{s \rightsquigarrow_{\mathrm{v}} s'}{C_{c,c}[s] \rightsquigarrow_{\mathrm{v}} C_{c,c}[s']}
$$

$$
\frac{v \rightsquigarrow_{\mathrm{v}} v'}{C_{c,v}[v] \rightsquigarrow_{\mathrm{v}} C_{c,v}[v']} \qquad \frac{s \rightsquigarrow_{\mathrm{v}} s'}{C_{v,c}[s] \rightsquigarrow_{\mathrm{v}} C_{v,c}[s']}
$$

$$
\frac{v \rightsquigarrow_{\mathrm{v}} v'}{C_{v,v}[v] \rightsquigarrow_{\mathrm{v}} C_{v,v}[v']}
$$

Figure 14. Strong CBV reduction

We fix two crucial properties of the eager let w.r.t. strong reduction.

Lemma 5.1. *The following hold:*

1. *Let* $M \rightsquigarrow^* M'$ *and* $N \rightsquigarrow^* N'$. *Then* $\mathbf{let}\ x \Leftarrow M\ \mathbf{in}\ N \rightsquigarrow^* \mathbf{let}\ x \Leftarrow M'\ \mathbf{in}\ N'$.
2. *Let* $N \rightsquigarrow^+ N'$ *and for all* V, V' *s.t.* $V \rightsquigarrow V'$, $N[V/x] \rightsquigarrow^+ N[V'/x]$. *Then* $\mathbf{let}\ x \Leftarrow M\ \mathbf{in}\ N \rightsquigarrow^+ \mathbf{let}\ x \Leftarrow M\ \mathbf{in}\ N'$.

5.1.1 Confluence

We show confluence of CBPV using the technique of Tait-Martin-Löf, refined by Takahashi [1989]. As the proof is completely analogous to its counterpart in the full λ-calculus, we give the definition of parallel reduction $\Rrightarrow$ and the parallel reduction function ϱ in the appendix [Forster et al. 2018] and only state the lemmas specific to CBPV for computations, while the analogous statements hold for values:

Lemma 5.2. $M \Rrightarrow M$

Lemma 5.3. $\rightsquigarrow$ *is included in* $\Rrightarrow$, *which is included in* $\rightsquigarrow^*$.

Lemma 5.4. $\Rrightarrow$ *is compatible with renamings and substitutions, explicitly for all renamings* ρ *and substitutions* σ_1, σ_2

1. *If* $M \Rrightarrow N$, *then* $M[\rho] \Rrightarrow N[\rho]$.
2. *If* $\forall x.\ \sigma_1\ x \Rrightarrow \sigma_2\ x$ *and* $M \Rrightarrow N$, *then* $M[\sigma_1] \Rrightarrow N[\sigma_2]$.

Lemma 5.5. *If* $M \Rrightarrow N$, *then* $N \Rrightarrow \varrho\ M$.

Theorem 5.6 (Confluence). *Strong reduction* $\rightsquigarrow$ *is confluent.*

Strong CBN reduction frames

$$\begin{aligned} C \;::=\;& s\,[\,] \mid [\,]\,s \mid \mathbf{case}([\,], x_1.t_1, x_2.t_2) \\ &\mid \mathbf{case}(s, x_1.[\,], x_2.t_2) \mid \mathbf{case}(s, x_1.t_1, x_2.[\,]) \\ &\mid \lambda x.[\,] \mid (s, [\,]) \mid ([\,], s) \mid \mathbf{inj}_b\,[\,] \mid \mathbf{prj}_b\,[\,] \end{aligned}$$

Strong CBN reduction $\boxed{M \rightsquigarrow_n M'}$

$$\frac{M \succ_n M'}{M \rightsquigarrow_n M'} \qquad \frac{M \rightsquigarrow_n M'}{C[M] \rightsquigarrow_n C[M']}$$

Figure 15. Strong CBN reduction

5.2 CBV

We define strong CBV reduction by extending the reduction frames to allow reductions in every context (fig. 14). Once again we write $\mathcal{X}_{v,v}, \mathcal{X}_{c,v}, \mathcal{X}_{v,c}, \mathcal{X}_{c,c}$ for the multi-layered versions of the strong reduction frames.

Again we aim for a simulation result. However, in contrast to the weak case, $s \Downarrow v \leftrightarrow \overline{s} \Downarrow \overline{v}$ cannot be proven. Both backward simulation and forward termination do not (and probably cannot) hold. This is because a strong CBPV semantics no longer provides the required fixed evaluation order: The translation can do more steps than CBV.

As an example, consider the (non well-typed) term

$$(\mathbf{val}\,(\lambda x.\mathbf{val}\,x))\;((\mathbf{val}\,z)\;(\mathbf{val}\,z)).$$

This term is obviously normal. However, its translation

$$\mathbf{let}\; x \leftarrow (z!\; z)\;\mathbf{in}\;\{\lambda z.\mathbf{return}\; z\}!\; x$$

can reduce first the force-thunk-pair and then do a beta reduction. While the force-thunk-pair could be eliminated in the translation, we conjecture that no translation can prevent this beta-reduction. With the same example, backward simulation fails to hold. However, we conjecture that the full simulation result does hold in the well-typed case, similar to [Levy 2006].

Forward simulation and backwards termination still hold and suffice to show strong normalisation.

Lemma 5.7.

1. *If $s \rightsquigarrow_v t$, then $\overline{s} \rightsquigarrow^+ \overline{t}$.*
2. *If $s \rightsquigarrow_v^* t$, then $\overline{s} \rightsquigarrow^* \overline{t}$.*

Corollary 5.8. *If $\overline{s}$ is normal, then s is normal.*

5.3 CBN

We define strong reduction $\rightsquigarrow_n$ for CBN in fig. 15. Note that this yields exactly the full λ-calculus with sums and products.

Substitutions are translated to $\overline{\sigma} := \lambda x. \begin{cases} y & \text{if } \sigma x = y \\ \{\overline{\sigma s}\} & \text{otherwise.} \end{cases}$

For strong reduction, deep force-thunk pairs can be reduced, and thus the translation is well-behaved w.r.t. substitution:

Lemma 5.9. *The following hold:*

1. $\overline{s}[\overline{\sigma}] \rightsquigarrow^* \overline{s[\sigma]}$.
2. *If $\sigma x! \rightsquigarrow^* \tau x!$ for all x, then $\overline{s}[\sigma] \rightsquigarrow^* \overline{s}[\tau]$.*

The forward direction then is straightforward:

Lemma 5.10. *The following hold:*

1. $\overline{s}[\{\overline{t}\}/x] \rightsquigarrow^* \overline{s[t/x]}$
2. *If $s \rightsquigarrow_n t$, then $\overline{s} \rightsquigarrow^+ \overline{t}$.*

We use the full simulation relation for the other direction:

Lemma 5.11. *Let $s \Mapsto_n M$.*

1. *$M \rightsquigarrow^* \overline{s}$ and $s \rightsquigarrow_n^* t$ for some t.*
2. *If $M \rightsquigarrow N$, then $s \rightsquigarrow_n^* t$ for some t with $t \Mapsto_n N$.*
3. *If $M \rightsquigarrow^* N$, then $s \rightsquigarrow_n^* t$ for some t with $t \Mapsto_n N$.*

Corollary 5.12. *If $\overline{s} \rightsquigarrow^* N$, then $N \rightsquigarrow^* \overline{t}$ and $s \rightsquigarrow_n^* t$ for some t.*

We can deduce a confluence proof for the full λ-calculus:

Theorem 5.13. *The full λ-calculus is confluent.*

Proof. Let $s \rightsquigarrow_n^* t_1$ and $s \rightsquigarrow_n^* t_2$. Then $\overline{s} \rightsquigarrow^* \overline{t_1}$ and $\overline{s} \rightsquigarrow^* \overline{t_2}$. Since CBPV is confluent, there is M s.t. $\overline{t_1} \rightsquigarrow^* M$ and $\overline{t_2} \rightsquigarrow^* M$.

There are now u_1 and u_2 with $t_1 \rightsquigarrow_n^* u_1$, $t_1 \rightsquigarrow_n^* u_2$, $u_1 \Mapsto_n M$ and $u_2 \Mapsto_n M$. Since $\Mapsto_n$ is injective $u_1 = u_2$. □

Similar to the weak case, we then recover the following results w.r.t. evaluation:

Theorem 5.14. *If $\overline{s}$ evaluates to M under $\rightsquigarrow$ then $M = \overline{t}$ and s evaluates to t under $\rightsquigarrow_n$.*

Lemma 5.15. *If s is normal, then $\overline{s}$ is normal.*

Theorem 5.16. *If s evaluates to t under $\rightsquigarrow_n$, then $\overline{s}$ evaluates to $\overline{t}$ under $\rightsquigarrow$.*

6 Strong Normalisation

We show strong CBPV normalisation analogous to the proof of weak normalisation, using strong semantic typing (fig. 16).

In order to show strong normalisation, we extend the semantic typing to open values ($V \in \mathcal{V}^{\circ}[A]$) by including variables and change the definition of the $\mathcal{E}[C]$ predicate. For weak normalisation, a computation M is in $\mathcal{E}[C]$ whenever it is weakly normalising and its normal form satisfies the $\mathcal{C}[C]$ predicate. In order to obtain strong normalisation, we extend the definition of $\mathcal{E}[C]$ to obtain guarantees for all reduction paths, as well as under arbitrary context extensions, e.g., renamings. Intuitively, we define $\mathcal{E}[C]$ to be the smallest (Kripke-) reducibility candidate containing $\mathcal{C}[C]$. As a slight simplification, it turns out that the closure under context extension is only relevant in the case of function types and so we include it into the definition of $\mathcal{C}[A \to C]$.

In addition, it is not sufficient for normal forms to satisfy the $\mathcal{C}[C]$ predicate. Instead, we need to have this property for all "active" computations. A computation M is *active*, if M is of the form $\langle\rangle$, **return** V, $\lambda x.M$, or $\langle M_1, M_2\rangle$.

The proof is similar to the proof of weak normalisation and consists of three parts that are all shown by straightforward structural inductions. First, we show generic properties

Value Semantic Typing $\boxed{V \in \mathcal{V}[A],\ V \in \mathcal{V}^\circ[A]}$

$$\mathcal{V}[0] := \emptyset \qquad \mathcal{V}[1] := \{()\}$$

$$\mathcal{V}[A_1 \times A_2] := \{(V_1, V_2) \mid V_1 \in \mathcal{V}^\circ[A_1], V_2 \in \mathcal{V}^\circ[A_2]\}$$

$$\mathcal{V}[A_1 + A_2] := \{\mathbf{inj}_i\ V \mid V \in \mathcal{V}^\circ[A_i]\}$$

$$\mathcal{V}[U\ C] := \{\{M\} \mid M \in \mathcal{E}[C]\}$$

$$\frac{}{x \in \mathcal{V}^\circ[A]} \qquad \frac{V \in \mathcal{V}[A]}{V \in \mathcal{V}^\circ[A]}$$

Computation Semantic Typing $\boxed{M \in \mathcal{C}[C]}$

$$\mathcal{C}[\top] := \{\langle\rangle\} \qquad \mathcal{C}[F\ A] := \{\mathbf{return}\ V \mid V \in \mathcal{V}^\circ[A]\}$$

$$\mathcal{C}[A \to C] := \{\lambda x.M \mid \forall \rho (V \in \mathcal{V}^\circ[A]).\ M[\rho[x := V]] \in \mathcal{E}[C]\}$$

$$\mathcal{C}[C_1 \,\&\, C_2] := \{(M_1, M_2) \mid M_1 \in \mathcal{E}[C_1], M_2 \in \mathcal{E}[C_2]\}$$

Semantic Typing

$$\frac{\begin{array}{c}\text{active } M \to M \in \mathcal{C}[C] \\ \forall N.\ M \rightsquigarrow N \to N \in \mathcal{E}[C]\end{array}}{M \in \mathcal{E}[C]}$$

$$\mathcal{G}[\Gamma] := \{\gamma \mid \forall (x : A) \in \Gamma, \gamma x \in \mathcal{V}^\circ[A]\}$$

$$\Gamma \vDash V : A := \forall \gamma \in \mathcal{G}[\Gamma].\ V[\gamma] \in \mathcal{V}^\circ[A]$$

$$\Gamma \vDash M : C := \forall \gamma \in \mathcal{G}[\Gamma].\ M[\gamma] \in \mathcal{E}[C]$$

Figure 16. CBPV strong semantic typing

of our definitions, i.e., that $\mathcal{E}[C]$ and $\mathcal{V}^\circ[A]$ are indeed reducibility candidates.

Lemma 6.1. *The following hold for strong semantic typing, and analogously for values:*

1. *If $M \in \mathcal{E}[C]$ then* SN $(\rightsquigarrow)$ M.
2. *If $M \in \mathcal{E}[C]$ and $M \rightsquigarrow N$ then $N \in \mathcal{E}[C]$.*
3. *If $M \in \mathcal{E}[C]$ then $M[\rho] \in \mathcal{E}[C]$.*

Lemma 6.2. *The following hold:*

1. id $\in \mathcal{G}[\Gamma]$.
2. *If $V \in \mathcal{V}^\circ[A]$ and $\sigma \in \mathcal{G}[\Gamma]$ then $\sigma[x := V] \in \mathcal{G}[\Gamma, x : A]$.*
3. *If $\sigma \in \mathcal{G}[\Gamma]$ then $\sigma[\rho] \in \mathcal{G}[\Gamma]$.*

In the second part we show that our definitions are compatible with the syntax of CBPV.

Lemma 6.3. *Semantic typing is compatible with the term structure of CBPV, for instance $M_1 \in \mathcal{E}[F\ A]$ and $M_2[V/x] \in \mathcal{E}[C]$ for all $V \in \mathcal{V}^\circ[A]$ imply* **let** $x \leftarrow M_1$ **in** $M_2 \in \mathcal{E}[C]$.

Using the compatibility lemmas it is then straightforward to show that syntactic typing implies semantic typing, from which strong CBPV normalisation follows.

Theorem 6.4. *If $\Gamma \vdash M : C$, then $\Gamma \vDash M : C$ and if $\Gamma \vdash V : A$, then $\Gamma \vDash V : A$.*

Corollary 6.5. *If $\Gamma \vdash M : C$, then $M \in \mathcal{E}[C]$ and if $\Gamma \vdash V : A$, then $V \in \mathcal{V}[A]$.*

Theorem 6.6. *The following hold:*

1. *Simply typed CBPV is strongly normalising.*
2. *Simply typed CBV is strongly normalising.*
3. *Simply typed CBN is strongly normalising.*

7 Observational Equivalence

In this section, we define observational equivalence [Mitchell 1996], written $\Gamma \vdash M \simeq N : C$, for CBPV, CBV, and CBN. Intuitively, two terms are *observationally equivalent* if they are computationally indistinguishable.

We prove that $\rightsquigarrow$ and thus $\leadsto$ are contained in observational equivalence, which will be crucial for the later adequacy proof. Furthermore, observational equivalence of the translation of two terms s, t immediately implies observational equivalence of s and t.

7.1 CBPV

Recall that we write $\mathcal{X}_{c,c}$ for contexts where the hole expects a computation and which returns computations. We define a context typing judgment $\Gamma[\Gamma'] \vdash \mathcal{X}_{c,c} : C[C']$ such that for all computation $\Gamma' \vdash M : C'$ we have $\Gamma \vdash \mathcal{X}_{c,c}[M] : C$, but do not spell it out in detail.

We call the following class of value types *ground*:

$$G \ := 0 \mid 1 \mid G_1 \times G_2 \mid G_1 + G_2$$

Ground types include $\mathbb{B} := 1 + 1$ with the elements **true** $:=$ $\mathbf{inj}_1\ ()$, **false** $:= \mathbf{inj}_0\ ()$.

Two terms $\Gamma \vdash M, N : C$ are *observationally equivalent* [Levy 2012], written $\Gamma \vdash M \simeq N : C$, if for all closed ground returner contexts $\mathcal{X}_{c,c}$ (i.e. $\emptyset[\Gamma] \vdash \mathcal{X}_{c,c} : F\,G[C]$) and values V:

$$\mathcal{X}_{c,c}[M] \leadsto^* \mathbf{return}\ V \quad \textit{iff} \quad \mathcal{X}_{c,c}[N] \leadsto^* \mathbf{return}\ V$$

The definition for values is analogous.

Lemma 7.1. $\simeq$ *is an equivalence relation on typed terms.*

Given the previous lemma, the only thing missing for closure of observational equivalence under $\rightsquigarrow$ is closure under primitive steps. This property is surprisingly intricate and not spelled out in detail in [Forster 2016; Forster et al. 2017]. Rizkallah et al. [2018] use eager normal form bisimulations to obtain it.

Here, we use a different approach via *logical equivalence* $\Gamma \vDash M \sim N : C$ (fig. 17) [Pitts 2005]. On typed terms, logical equivalence is an equivalence relation relating computationally indistinguishable terms. We show that this relation can be embedded between reductions and observational equivalence. While these results could be obtained using the purely syntactic methods presented in [Crary 2009], the logical equivalence will allow us to establish that observational equivalence also contains η-reductions and let-sequencing.

In its essence, the logical equivalence is similar to the logical relations of syntactic typing. The key difference is

$$\textbf{\textit{Value Relation}}\ \boxed{(V,W)\in\mathcal{V}[A]}$$

$$\mathcal{V}[0] := \emptyset \qquad \mathcal{V}[1] := \{((),())\} \qquad \mathcal{V}[A_1\times A_2] :=$$

$$\{((V_1,V_2),(W_1,W_2)) \mid (V_1,W_1)\in\mathcal{V}[A_1],(V_2,W_2)\in\mathcal{V}[A_2]\}$$

$$\mathcal{V}[A_1+A_2] := \{(\mathbf{inj}_i\,V,\mathbf{inj}_i\,W) \mid (V,W)\in\mathcal{V}[A_i]\}$$

$$\mathcal{V}[U\,C] := \{(\{M\},\{N\}) \mid (M,N)\in\mathcal{E}[C]\}$$

$$\textbf{\textit{Computation Relation}}\ \boxed{(M,N)\in\mathcal{C}[C]}$$

$$\mathcal{C}[\top] := \{(\langle\rangle,\langle\rangle)\}$$

$$\mathcal{C}[F\,A] := \{(\mathbf{return}\ V,\mathbf{return}\ W) \mid (V,W)\in\mathcal{V}[A]\} \qquad \mathcal{C}[A\to C] :=$$

$$\{(\lambda x.M,\lambda y.N) \mid \forall (V,W)\in\mathcal{V}[A].\ (M[V/x],N[W/y])\in\mathcal{E}[C]\}$$

$$\mathcal{C}[C_1\,\&\,C_2] :=$$

$$\{(\langle M_1,M_2\rangle,\langle N_1,N_2\rangle) \mid (M_1,N_1)\in\mathcal{E}[C_1],(M_2,N_2)\in\mathcal{E}[C_2]\}$$

$$\textbf{\textit{Logical Equivalence}}$$

$$\mathcal{E}[C] := \{(M,N) \mid \exists M'N'.\ M\Downarrow M' \text{ and } N\Downarrow N' \text{ and } (M',N')\in\mathcal{C}[C]\}$$

$$\mathcal{G}[\Gamma] := \{(\gamma_1,\gamma_2) \mid \forall (x:A)\in\Gamma.\ (\gamma_1 x,\gamma_2 x)\in\mathcal{V}[A]\}$$

$$\Gamma\vDash V\sim W : A := \forall(\gamma_1,\gamma_2)\in\mathcal{G}[\Gamma].\ (V[\gamma_1],W[\gamma_2])\in\mathcal{V}[A]$$

$$\Gamma\vDash M\sim N : C := \forall(\gamma_1,\gamma_2)\in\mathcal{G}[\Gamma].\ (M[\gamma_1],N[\gamma_2])\in\mathcal{E}[C]$$

Figure 17. CBPV logical equivalence

that the predicates now relate types with pairs of terms having computationally indistinguishable normal forms.

The following lemmas simplify reasoning about logical equivalence. The relation $\mathcal{E}[C]$ is closed under both expansion and reduction.

Lemma 7.2. *Logical equivalence and the relations $\mathcal{V}[A]$, $\mathcal{C}[C]$, $\mathcal{E}[C]$, and $\mathcal{G}[\Gamma]$ are symmetric and transitive.*

Lemma 7.3. *Let $M \rightsquigarrow^* M'$ and $N \rightsquigarrow^* N'$. Then $(M',N')\in\mathcal{E}[C]$ iff $(M,N)\in\mathcal{E}[C]$.*

We can extend substitutions with semantically equivalent values.

Lemma 7.4. *If $(\gamma_1,\gamma_2)\in\mathcal{G}[\Gamma]$ and $(V_1,V_2)\in\mathcal{V}[A]$ then $(\gamma_1[x:=V_1],\gamma_2[x:=V_2])\in\mathcal{G}[\Gamma,x:A]$.*

When reasoning about equivalent terms in evaluation position, we may reason about normal forms in these positions.

Lemma 7.5. *Let C_1,C_2 be weak reduction frames. If $(M,N)\in\mathcal{E}[C]$ and $\forall (M',N')\in\mathcal{C}[C].\ (C_1[M'],C_2[N'])\in\mathcal{E}[C']$, then $(C_1[M],C_2[N])\in\mathcal{E}[C']$.*

Lemma 7.6.

1. *$\mathcal{C}[C]\subseteq\mathcal{E}[C]$.*
2. *For all ground types G, if $(V,W)\in\mathcal{V}[G]$, then $V=W$.*

We proceed by proving that every well typed term is logically equivalent to itself and logical equivalence is a congruence relation. Both properties heavily rely on congruence lemmas for each syntactic construct and also hold for values.

Lemma 7.7 (Fundamental Property). *If $\Gamma\vdash M : C$, then $\Gamma\vDash M\sim M : C$.*

Lemma 7.8 (Congruence). *Logical equivalence is a congruence relation w.r.t. typed CBPV contexts.*

Using the fundamental property and congruence, we can finally deduce:

Lemma 7.9. *Logical equivalence contains $\succ$, $\rightsquigarrow$, and $\leadsto$. For example if $\Gamma\vdash M : C$ and $M\succ M'$ then $\Gamma\vDash M\sim M' : C$.*

This suffices to show that logical equivalence is sound w.r.t. observational equivalence:

Theorem 7.10. *If $\Gamma\vdash M,N : C$ and $\Gamma\vDash M\sim N : C$, then $\Gamma\vdash M\simeq N : C$.*

Proof. Let $\emptyset[\Gamma]\vdash\mathcal{X}_{c,c} : F\,G[C]$. By Lemma 7.8 we have $\emptyset\vDash\mathcal{X}_{c,c}[M]\sim\mathcal{X}_{c,c}[N] : F\,G$ for some ground type G. Hence $\mathcal{X}_{c,c}[M]\Downarrow\mathbf{return}\ V$ and $\mathcal{X}_{c,c}[N]\Downarrow\mathbf{return}\ V'$ for some $(V,V')\in\mathcal{V}[G]$. With Lemma 7.6 it follows that $V=V'$. By symmetry, it suffices to consider only one direction. Assume $\mathcal{X}_{c,c}[M]\rightsquigarrow^*\mathbf{return}\ W$. The claim follows with functionality of $\Downarrow$. □

Corollary 7.11. *Observational equivalence contains $\succ$, $\rightsquigarrow$, and $\leadsto$.*

Proof. By Lemma 7.9 and Theorem 7.10. □

7.2 CBV

We now define observational equivalence for CBV and prove that it can be established using the translation to CBPV.

We define $\mathbb{B}_v := 1+1$ with the two values $\mathbf{true}_v := \mathbf{inj}_1\,()$, $\mathbf{false}_v := \mathbf{inj}_0\,()$. Note that the translation coincides with the CBPV counterparts. We follow Levy [2012] and define for two terms $\Gamma\vdash s,t : A$ that $s\simeq_v t$ if for all closed boolean contexts $\emptyset[\Gamma]\vdash\mathcal{X}_{c,c} : \mathbb{B}_v[A]$:

$$\mathcal{X}_{c,c}[s]\rightsquigarrow_v^*\mathbf{val}\,\mathbf{true}_v \quad \textit{iff} \quad \mathcal{X}_{c,c}[t]\rightsquigarrow_v^*\mathbf{val}\,\mathbf{true}_v$$

The definition for values is analogous.

Lemma 7.12. *$\simeq_v$ is an equivalence relation on typed terms.*

Lemma 7.13. *If $\overline{s}\simeq\overline{t}$, then $s\simeq_v t$.*

Proof. For this proof we require a translation function $\overline{\mathcal{X}_{c,c}}$ on contexts s.t. $\overline{\mathcal{X}_{c,c}[s]}=\overline{\mathcal{X}_{c,c}}[\overline{s}]$ holds. However, in the de Bruijn representation we are using in Coq this would mean that we need to extend contexts by renamings and a version of the eager let construct. To circumvent this, we define a logically equivalent translation $\widehat{s}$ that avoids both problems. We give an exemplary case:
$\widehat{s\,t}=(\lambda x_1x_2.\mathbf{let}\ y_1\leftarrow x_1!\ \mathbf{in}\ \mathbf{let}\ y_2\leftarrow x_2!\ \mathbf{in}\ y_1!\ y_2)\ \{\widehat{s}\}\ \{\widehat{t}\}$
This definition can be lifted to a function $\widehat{\mathcal{X}_{c,c}}$ s.t. $\widehat{\mathcal{X}_{c,c}[s]}=\widehat{\mathcal{X}_{c,c}}[\widehat{s}]$.

For the proof it suffices to show one direction by symmetry of $\simeq$. Let $\overline{s}\simeq\overline{t}$ and $\mathcal{X}_{c,c}[s]\rightsquigarrow_v^*\mathbf{val}\,\mathbf{true}_v$. By Lemma 3.3 we have $\overline{\mathcal{X}_{c,c}[s]} \leadsto^* \overline{\mathbf{val}\,\mathbf{true}_\mathrm{v}}$. With the logical equivalence we obtain $\widehat{\mathcal{X}_{c,c}}[\widehat{s}] = \widehat{\mathcal{X}_{c,c}[s]} \leadsto^* \widehat{\mathbf{val}\,\mathbf{true}_\mathrm{v}}$. With Theorem 7.10 we obtain $\widehat{s} \simeq \widehat{t}$, thus by definition $\widehat{\mathcal{X}_{c,c}[t]} = \widehat{\mathcal{X}_{c,c}}[\widehat{t}] \leadsto^*$ $\widehat{\mathbf{val}\,\mathbf{true}_\mathrm{v}}$ since $\widehat{\mathbf{val}\,\mathbf{true}_\mathrm{v}} = \overline{\mathbf{val}\,\mathbf{true}_\mathrm{v}} = \mathbf{return}\ \mathbf{true}$ is ground. With the logical equivalence we deduce $\overline{\mathcal{X}_{c,c}[t]} \leadsto^* \overline{\mathbf{val}\,\mathbf{true}_\mathrm{v}}$. The claim follows with Theorem 3.9. □

7.3 CBN

For CBN, the situation is slightly different. The definition of $\mathbb{B}_\mathrm{n} := 1 + 1$ has more than two normal forms w.r.t. $\leadsto_n$, for instance $\mathbf{true}_\mathrm{n} := \mathbf{inj}_1\ ()$, $\mathbf{false}_\mathrm{n} := \mathbf{inj}_0\ ()$ and $\mathbf{inj}_0\ ((\lambda x.x)\ ())$. Only up to $\rightsquigarrow_n$ there are exactly two normal forms. This hints that defining observational equivalence for all boolean contexts in the same way as for CBV would be wrong, because $\rightsquigarrow_n$ would not be included anymore.

Levy [2012] circumvents this problem with a primitive type $\mathbb{B}_\mathrm{n}$. We use the definition of $\mathbb{B}_\mathrm{n}$ given above, but will alter the contexts in the definition of $\simeq_\mathrm{n}$ to ensure the existence of exactly two possible normal forms under $\leadsto_n$.

We define for two terms $\Gamma \vdash s, t : A$ that $s \simeq_\mathrm{n} t$ if for all closed boolean contexts $\emptyset[\Gamma] \vdash \mathcal{X} : \mathbb{B}_\mathrm{n}[A]$:

$$\mathbf{case}(\mathcal{X}[s], x.\mathbf{true}_\mathrm{n}, x.\mathbf{false}_\mathrm{n}) \leadsto_n^* \mathbf{true}_\mathrm{n}$$
$$\textit{iff}$$
$$\mathbf{case}(\mathcal{X}[t], x.\mathbf{true}_\mathrm{n}, x.\mathbf{false}_\mathrm{n}) \leadsto_n^* \mathbf{true}_\mathrm{n}$$

We can then obtain the expected results:

Lemma 7.14. *$\simeq_n$ is an equivalence relation on typed terms.*

Lemma 7.15. *If $\overline{s} \simeq \overline{t}$, then $s \simeq_n t$*

Proof. Analogous to the proof of Lemma 7.13. We again require a translation $\widehat{s}$ avoiding renamings and eager lets. We define $K := \mathbf{let}\, x \leftarrow [\ \ ]\, \mathbf{in}\, \mathbf{case}(x, \_.\mathbf{return}\ \mathbf{true}, \_.\mathbf{return}\ \mathbf{false})$ s.t. $K[\widehat{\mathbf{case}(\mathcal{X}, x.\mathbf{true}_\mathrm{n}, x.\mathbf{false}_\mathrm{n})}][\widehat{s}] \leadsto^* \mathbf{return}\ \mathbf{true}$ if and only if $\mathbf{case}(\mathcal{X}[s], x.\mathbf{true}_\mathrm{n}, x.\mathbf{false}_\mathrm{n}) \leadsto_n^* \mathbf{true}_\mathrm{n}$. □

8 Equational Theory

Levy [2006] gives a sound equational theory for CBPV with β, η and sequencing laws. Rizkallah et al. [2018] formalise soundness of an equational theory for CBPV with β-laws. We prove soundness for both theories w.r.t logical equivalence, concluding soundness w.r.t. observational equivalence.

For our version of CBPV the definition in [Rizkallah et al. 2018] coincides with the equivalence closure of strong reduction. We denote the equivalence closure of $\rightsquigarrow$, $\rightsquigarrow_n$, and $\rightsquigarrow_\mathrm{v}$ with $\equiv$, $\equiv_\mathrm{n}$, and $\equiv_\mathrm{v}$ respectively. We can easily prove that $\equiv$ is sound w.r.t. logical equivalence:

Lemma 8.1. *The following hold:*

1. *If $\Gamma \vdash M, N : C$ and $M \equiv N$, then $\Gamma \vDash M \sim N : C$.*
2. *If $s \equiv_v t$, then $\overline{s} \equiv \overline{t}$*
3. *If $s \equiv_n t$, then $\overline{s} \equiv \overline{t}$*

Using Lemma 7.10 we obtain soundness of $\equiv$ w.r.t $\simeq$:

Corollary 8.2. *If $M \equiv N$, then $M \simeq N$ for well-typed M, N.*

Additionally we can obtain soundness proofs of strong equivalence for typed CBV and CBN terms using the simulation results:

Corollary 8.3. *For all well-typed terms s, t:*

1. *If $s \equiv_v t$, then $s \simeq_v t$.*
2. *If $s \equiv_n t$, then $s \simeq_n t$.*

Levy [2006] proves soundness using adequacy of his denotational semantics. We can also prove his extended equational theory sound using logical equivalence for our setting:

Lemma 8.4. *The following equations can be proven without reference to weak normalisation:*

1. $\Gamma \vDash \{V!\} \sim V : U\,A$
2. $\Gamma \vDash M \sim \mathbf{let}\ x \leftarrow M\ \mathbf{in\ return}\ x : F\,A$
3. $\Gamma \vDash M[V/x] \sim \mathbf{case}(V, y.M[\mathbf{inj}_0\, y/x], z.M[\mathbf{inj}_1\, z/x]) : C$
4. $\Gamma \vDash M[V/x] \sim \mathbf{split}(V, y.z.M[(y,z)/x]) : C$
5. $\Gamma \vDash \mathbf{let}\ x \leftarrow (\mathbf{let}\ y \leftarrow M_1\ \mathbf{in}\ M_2)\ \mathbf{in}\ M_3 \sim$
   $\mathbf{let}\ x \leftarrow M_1\ \mathbf{in}\ (\mathbf{let}\ y \leftarrow M_2\ \mathbf{in}\ M_3) : C.$

Lemma 8.5. *The following equations can be proven* with *reference to weak normalisation:*

1. $\Gamma \vDash M \sim \lambda x.M\,x : A \to B$
2. $\Gamma \vDash M \sim \langle \mathbf{prj}_0\, M, \mathbf{prj}_1\, M \rangle : A_1\ \&\ A_2$
3. $\Gamma \vDash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ \lambda y.N \sim \lambda y.\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : A \to B$

We can also prove more useful equations not mentioned in [Levy 2006]:

Lemma 8.6. *The following hold:*

1. $\Gamma \vDash (\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N)V \sim \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ (N V) : C$
2. $\Gamma \vDash \mathbf{case}(V, y.\lambda x.M, y.\lambda x.N) \sim \lambda x.\mathbf{case}(V, y.M, y.N) :$ $A \to C$

9 Denotational Semantics

Giving denotational semantics amounts to giving a function $[\![\cdot]\!]$ assigning mathematical meaning to computational objects. A denotational semantics $[\![\cdot]\!]$ is adequate if $[\![M]\!] = [\![N]\!]$ implies $M \simeq N$. We define a set/algebra denotational semantics for CBPV, prove adequacy following the presentation in [Forster 2016], and use the translation to obtain adequate algebra semantics for CBV and CBN.

The denotation of value types are types in Coq, for instance $[\![A_1 \times A_2]\!] := [\![A_1]\!] \times [\![A_2]\!]$. The semantics for computation types is given for arbitrary monads T. The denotation of computation types are T-algebras, where a T-algebra C is a pair $(|C|, c : T|C| \to |C|)$ with $|C|$ being the carrier type of the algebra and c has certain properties with respect to the monad T. Algebras are closed under products, exponents and admit a trivial algebra, used as denotations for computational products, functions and unit respectively. For every type A one can construct a free T-algebra $F_T A$ and define $[\![F\,A]\!] := F_T\,[\![A]\!]$. Furthermore, $[\![U\,C]\!] := |\,[\![C]\!]\,|$.

Contents	Spec	Proofs
Setup	350	250
Translating CBV and CBN to CBPV	1250	500
Weak Reduction	200	450
Weak Normalisation	100	150
Strong Reduction	800	950
Strong Normalisation	200	300
Observational Equivalence	250	350
Equational Theory	100	200
Denotational Semantics	700	600
Total	3,950	3,750

Figure 18. Overview of the code

The adequacy proof uses logical relations R_A and R_C. R_A relates elements of $[\![A]\!]$ with closed values of type A. R_C relates elements of $|\,[\![C]\!]\,|$ with closed computations of type C. We only outline the structure of the adequacy proof here and omit some auxiliary statements, all statements concerning value and the definition of $[\![\cdot]\!]$ for terms. For full definitions and proofs, we refer to [Forster 2016] and the Coq development. Once more all statements hold for values whenever applicable.

Lemma 9.1. *If $[\![M]\!] = [\![N]\!]$ then $[\![\mathcal{X}[M]]\!] = [\![\mathcal{X}[N]]\!]$.*

Lemma 9.2. *If $(a, M) \in R_C$ and $M \simeq N$ then $(a, N) \in R_C$.*

We state the basic lemma only for closed computations, but it has to be proven in a more general form:

Lemma 9.3 (basic lemma). *If $\vdash M : C$ then $([\![M]\!], M) \in R_C$.*

Theorem 9.4. *The denotational semantics for CBPV is adequate. Explicitly, for every monad where* **return** *is injective, $\Gamma \vdash M, N : C$ s.t. $[\![M]\!] = [\![N]\!]$ we have $\Gamma \vdash M \simeq N : C$.*

We lift denotations from CBPV to CBN/CBV in the obvious way and obtain an algebra semantics for CBN and a set/algebra semantics CBV:

$$[\![s]\!]_{\mathrm{v}} := [\![\bar{s}]\!] \qquad\qquad [\![s]\!]_{\mathrm{n}} := [\![\tilde{s}]\!]$$

Adequacy of this denotational semantics immediately follows:

Corollary 9.5. *For all closed, well typed expressions s, t, if $[\![s]\!]_v = [\![t]\!]_v$, then $s \simeq_v t$*

Corollary 9.6. *For all closed, well typed expressions s, t, if $[\![s]\!]_n = [\![t]\!]_n$, then $s \simeq_n t$*

10 Formalisation and Autosubst 2

Our development consists out of roughly 8000 lines of code, with about 55% specification and 45% proofs. Figure 18 in the appendix depicts the code distribution.

The development does not use advanced automation, but basic automation provided by Coq and some simple custom tactics. Coq's built-in automation proved useful for minor changes in definitions, leading to only minor changes in proofs. Especially in the verification of the translations, Coq treats many trivial cases automatically, allowing the user to focus on the interesting parts. Overall, we think that the Coq formalisation is similar to proofs on paper: What is hard in Coq is also not easy on paper, and vice versa. We think that providing a formalisation to this extent in a proof assistant without inductive types and tactics would be a considerably bigger effort, if not entirely infeasible.

Autosubst 2 Although we use a named representation of syntax in the paper, we use well-scoped de Bruijn indices in the Coq development. If one is familiar with de Bruijn, the translations are straightforward and can be easily retraced using the Coq links. Well-scoped syntax proved very useful at this stage: In subtle cases, it supports the user in choosing the right definitions and re-thinking the needed liftings.

The definitions of the syntax for CBPV, CBV and CBN were generated by the Autosubst 2 framework [Stark et al. 2018]. Autosubst 2 takes a HOAS specification as input and outputs definitions using well-scoped de Bruijn syntax and a small library containing standard lemmas regarding renamings and substitution. In our case this means that treating binders becomes a non-issue: many challenges that previously came with treating binders in formalisation are taken care of by the respective Autosubst 2 tactics. Together with choosing the correct statements (e.g. context renaming and context morphism lemmas as also stated in this paper), we could observe no overhead in the number of lemmas.

We discovered two possible extensions of the Autosubst tool: First, we had to prove injectivity of renamings by hand. This could be taken care of by an automated proof coming as part of the generated library. Second, we had to define the translations from CBN/CBV on the concrete de Bruijn syntax and prove compatibility with substitution and renaming by hand. If Autosubst would incorporate the techniques from [Kaiser et al. 2018; Schäfer and Stark 2018], both the translations and all type systems could be defined on the HOAS level, with some properties deduced automatically.

Coq. One key technique for our formalisation is setoid rewriting [Sozeau 2010], especially for the verification of the translations. Setoid rewriting allows us to rewrite expressions using arbitrary congruences. However, setoid rewriting with the transitive closure (e.g. $\leadsto^+$) lacks a satisfactory solution and we have to define custom (but general) Ltac tactics.

We formalise multi-level contexts for CBPV in Coq as a mutual inductive family of two types with a parameter indicating the type of the hole of the context. This made the definition slightly more concise than a mutual inductive definition with four types.

For Lemma 7.15 we could have used renamings in contexts, implemented by an additional constructor to the context type. To avoid this, we use the logically equivalent definitions of translation not based on renamings.

Denotation functions for terms are defined on the derivation of their typing judgement. Thus, we have to lift the typing judgement from Coq's impredicative propositional type `Prop` into a predicative universe `Type`. We would then need mutual induction principles with strong type-theoretic connectives (i.e. $\times$ instead of $\wedge$), which Coq in its present version (8.8) cannot synthesise. Stating the principle by hand is cumbersome for a syntax with many syntactic constructs. We circumvent this problem by giving proofs as fixed-points instead of applying the induction principle. Fortunately, the upcoming Coq 8.9 will be able to automatically generate these principles.

Conclusion. In total, we think that developing the metatheory of CBPV and the translation to CBN/CBV as we did *without* a proof assistant would have been *more* work. The technically involved work requires a lot of intuition for CBPV and there are many side conditions. Some of them are trivial and can be resolved by Coq automatically, while some only seem trivial on paper, but depend on crucial details that have to be worked out.

11 Related Work

Certainly the most complete discussion of the operational theory of CBPV is given by Levy [2012]. Levy proves the simulation results we have, but for typed terms and w.r.t big-step semantics. He does not consider strong reduction. Levy [2006] also considers an equational theory including β, η and commuting lets and proves soundness w.r.t a categorical denotational semantics.

Doczkal and Schwinghammer [2007] consider simply-typed CBPV with arbitrary sums and products. They extend the usual reduction by η-rules and permuting conversions for let, making reduction non-deterministic. They prove strong normalisation for their reduction using $\top\top$-lifting for logical relations, but unlike our development, their reduction is not confluent.

Rizkallah et al. [2018] formalise the soundness of an equational theory (with β-laws) w.r.t. observational equivalence (which they call contextual equivalence) for CBPV in Coq. They consider untyped CBPV with binary sums and with a mutually-recursive letrec construct. Their formalisation uses de Bruijn indices and a single datatype for both values and computations. Since we do not consider letrec, we work with a mutual inductive definition of values and computations, which we found pleasant. Their soundness proof uses normal form bisimulation as an intermediate notion. They do not formalise contexts as first-class objects and rely on conditional compatible closures to mimic contexts. We found our soundness proof via logical equivalence to be pleasantly semantic. Treating contexts explicitly in Coq is about as hard as treating them in all detail on paper: the definition is unwieldy, but nothing complicated happens.

Forster et al. [2017] extend CBPV with algebraic effects, monadic reflection and delimited control to express computational effects and give translations between the different systems. Our adequacy proofs follows their setup, outlined in more detail in [Forster 2016]. They also prove normalisation for ground returners, but not for terms of arbitrary type. Their progress and preservation proofs and the correctness of the translations are verified in Abella.

Crary [2009] proves confluence and soundness of the equational theory for an untyped CBV λ-calculus using the method from Takahashi [1989]. We think that his method could be adapted to CBPV and would yield proofs of a similar complexity, also covering untyped terms.

12 Future Work

The simple type system for CBPV we consider enforces normalisation and thus does not allow arbitrary recursion. CBPV can be extended by a syntactic construct to allow for recursion (see e.g. [Rizkallah et al. 2018]) and the type system can be extended accordingly. We conjecture that logical equivalence can be changed s.t. proving soundness of the β-equations and the equations in Lemma 8.4 works with basically unchanged proofs.

CBPV was conceived as an idealised calculus well-suited for the inclusion of effects. We want to formalise a version of CBPV with effects, both by inclusion of native effects and algebraic effect handlers. On paper, proofs over extended CBPV, as for instance done in [Forster et al. 2017], leave out the cases that have been treated before the extension. When working in a proof assistant, this is not an option. We want to investigate how well such extensions can be treated in Coq without code duplication.

As a third possibility, CBPV can be extended with polymorphism. We conjecture that our proof of strong normalisation using Kripke logical relation extends to this case. Since the simulation results we prove hold for untyped terms, we conjecture that they can also be used to obtain e.g. strong normalisation for System F directly.

Parts of this paper have been formalising the first sections of [Forster et al. 2017], which comes with a partial Abella formalisation for the operational parts. We want to formalise the remaining results in Coq, which would be an interesting case study in comparing Coq to Abella, similar to [Kaiser et al. 2017], but in a more operational spirit.

Lastly, it would be interesting to analyse how the full equational theory from Levy [2006] including η-laws and commuting lets applies to compiler verification, e.g. to verify let-floating optimisation [Peyton Jones et al. 1996].

Acknowledgments

We want to thank Paul Levy for discussion and pointers to related work, Ohad Kammar for LaTeX macros for typesetting CBPV, and the anonymous reviewers for their suggestions.

References

Coq proof assistant. 2018. http://coq.inria.fr.

Karl Crary. 2009. A simple proof of call-by-value standardization. *Computer Science Department (2009)* 474 (2009).

Christian Doczkal and Jan Schwinghammer. 2007. A Proof of Strong Normalization for Call-by-push-value. Unpublished note. http://www.ps.uni-saarland.de/Publications/details/Doczkal:Schwinghammer:07.html

Derek Dreyer, Ralf Jung, Jan-Oliver Kaiser, Hoang-Hai Dang, and David Swasey. 2018. Semantics of Type Systems – Lecture Notes. (2018).

Yannick Forster. 2016. On the expressive power of effect handlers and monadic reflection. MPhil thesis, University of Cambridge. (2016). https://www.ps.uni-saarland.de/~forster/downloads/mphil-thesis.pdf

Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. 2017. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *Proceedings of the ACM on Programming Languages* 1, ICFP (2017), 13.

Yannick Forster, Steven Schäfer, Simon Spies, and Kathrin Stark. 2018. Appendix for Call-By-Push-Value in Coq: Operational, Equational, and Denotational Theory. https://ps.uni-saarland.de/cbpv-in-coq

J-Y Girard. 1971. Une extension de l'interpretation de Godel a l'analyse et son application a l'elimination des coupures dans l'analyse et la theorie des types. In *Proc. 2nd Scandinavian Logic Symp.* North-Holland, 63–92.

Jonas Kaiser, Brigitte Pientka, and Gert Smolka. 2017. Relating System F and Lambda2: A Case Study in Coq, Abella and Beluga. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 84. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Jonas Kaiser, Steven Schäfer, and Kathrin Stark. 2018. Binder aware recursion over well-scoped de Bruijn syntax. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, 293–306.

Roland Leißa, Marcel Köster, and Sebastian Hack. 2015. A Graph-Based Higher-Order Intermediate Representation. In *International Symposium on Code Generation and Optimization*.

Paul Blain Levy. 1999. Call-by-push-value: A subsuming paradigm. In *International Conference on Typed Lambda Calculi and Applications*. Springer, 228–243.

Paul Blain Levy. 2006. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation* 19, 4 (2006), 377–414.

Paul Blain Levy. 2012. *Call-by-push-value: A Functional/imperative Synthesis*. Vol. 2. Springer Science & Business Media.

John C Mitchell. 1996. *Foundations for programming languages*. Vol. 1. MIT press Cambridge.

Simon Peyton Jones, Will Partain, and André Santos. 1996. Let-floating: moving bindings to give faster programs. In *ACM SIGPLAN Notices*, Vol. 31. ACM, 1–12.

Andrew Pitts. 2005. Typed operational reasoning. *Advanced Topics in Types and Programming Languages* (2005), 245–289.

Christine Rizkallah, Dmitri Garbuzov, and Steve Zdancewic. 2018. A Formal Equational Theory for Call-By-Push-Value. In *International Conference on Interactive Theorem Proving*. Springer, 523–541.

Steven Schäfer and Kathrin Stark. 2018. Embedding Higher-Order Abstract Syntax in Type Theory. In *Abstract for Types Workshop*.

Matthieu Sozeau. 2010. A new look at generalized rewriting in type theory. *Journal of formalized reasoning* 2, 1 (2010), 41–62.

Kathrin Stark, Schäfer Steven, and Jonas Kaiser. 2018. Autosubst 2: Reasoning with Multi-Sorted de Bruijn Terms and Vector Substitutions. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*.

William W Tait. 1967. Intensional interpretations of functionals of finite type I. *The journal of symbolic logic* 32, 2 (1967), 198–212.

Masako Takahashi. 1989. Parallel reductions in λ-calculus. *Journal of Symbolic Computation* 7, 2 (1989), 113–123.