The Call-By-Value Lambda-Calculus with Generalized Applications

José Espírito Santo
Centre of Mathematics, University of Minho, Portugal
jes@math.uminho.pt

Abstract

The lambda-calculus with generalized applications is the Curry-Howard counterpart to the system of natural deduction with generalized elimination rules for intuitionistic implicational logic. In this paper we identify a call-by-value variant of the system and prove confluence, strong normalization, and standardization. In the end, we show that the cbn and cbv variants of the system simulate each other via mappings based on extensions of the "protecting-by-a-lambda" compilation technique.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Lambda calculus

Keywords and phrases Generalized applications, Natural deduction, Strong normalization, Standardization, Call-by-name, Call-by-value, Protecting-by-a-lambda

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.35

Funding The author was supported by Fundação para a Ciência e a Tecnologia (FCT) through project UID/MAT/00013/2013.

1 Introduction

The λ-calculus with generalized applications [8], named system ΛJ, or λJ-calculus, corresponds, under the Curry-Howard isomorphism, to the system of natural deduction with generalized elimination rules [10, 16], in the setting of intuitionistic implicational logic. As a variant of the λ-calculus, ΛJ can be qualified naively as being a call-by-name (cbn) system, simply because its β-rule prescribes that, in functional application, functions are called without prior evaluation of arguments. In this paper we propose a call-by-value (cbv) variant of system ΛJ and prove some of its properties. With this, we develop the cbv side of natural deduction.

The novel system is named system $\Lambda J_{\mathtt{v}}$, or the $\lambda J_{\mathtt{v}}$-calculus. Notably, the syntax of proof terms remains the same as that of system ΛJ - and our purpose is to define cbv reduction rules appropriate for this syntax. Moreover, the reduction rules of $\Lambda J_{\mathtt{v}}$ will look like those of ΛJ: there is a β-rule (corresponding to a "detour" conversion rule) and a π (corresponding to a commuting conversion rule), the latter in fact unchanged w.r.t. ΛJ; the notion of β-redex is also unchanged; the only thing that will change is the concept of substitution.

Plotkin [11] sets a criterion for a calculus to be qualified as call-by-value: it should enjoy a standardization theorem, and the notion of standard reduction sequence should be based on a notion of call-by-value evaluation. We will prove a standardization theorem for $\Lambda J_{\mathtt{v}}$ that makes an explicit link to a notion of cbv evaluation. We also prove the main rewriting-theoretic properties: confluence and strong normalization.

Plotkin [11] shows that cbn and cbv calculi based on the syntax of ordinary λ-terms simulate each other via cps translations. The need to resort to cps translations is justified in [11] by the fact that the "protecting-by-a-lambda" compilation technique (which easily gives a simulation of cbn by cbv) does not extend to a simulation in the opposite direction. Here we show that, when the syntax allows generalized applications, cps translations are not needed to obtain simulation in both ways, as both simulations can be based on "protecting-by-a-lambda".

Plan of the paper. Section 2 reviews Λ, Plotkin's $\Lambda_{\mathtt{v}}$, and ΛJ. Section 3 introduces $\Lambda J_{\mathtt{v}}$. Sections 4 and 5 prove strong normalization and confluence, respectively. Section 6 proves standardization. Section 7 simulates the cbn and cbv variants into each other. Section 8 summarizes our contributions and concludes.

2 Background

System Λ. The ordinary λ-calculus is denoted Λ. The λ-terms are given by:

$$\text{(Terms) } M,N,P,Q ::= V \mid MN \qquad \text{(Values) } V,W ::= x \mid \lambda x.M$$

We work modulo α-equivalence and assume silent renaming of bound variables as needed. We write $x \notin M$ to say that x does not occur free in M. Substitution is denoted $[N/x]M$.

Λ is equipped with the β-rule $(\lambda x.M)N \to [N/x]M$, which generates the relation $\to_\beta$ - the compatible closure of β. We use the common notations $\to_\beta^=$, $\to_\beta^+$, $\to_\beta^*$, and $=_\beta$ for, respectively, the reflexive, transitive, reflexive-transitive, and equivalence closures of $\to_\beta$ (and similarly for any other reduction rule in any other calculus in this paper).

An important role is played by λ-terms with *holes*. A hole, denoted $[\cdot]$, is a special place-holder that can be *filled* with a λ-term. We will only consider terms with a single hole, and call them *contexts*. If $\mathcal{C}$ is a context, the $\mathcal{C}[M]$ denotes the λ-term resulting from filling the single hole of $\mathcal{C}$ with M. Notice contexts allow an alternative definition of $\to_\beta$: $P \to_\beta Q$ iff $P = \mathcal{C}[(\lambda x.M)N]$ and $Q = \mathcal{C}[[N/x]M]$, for some $\mathcal{C}$.

We consider the familiar, Curry-style typing system, for assigning simple types to λ-terms. Types, ranged over by A, B, C, etc. are formulas of implicational logic, which can either be an atom p or an implication $A \supset B$. A sequent is an expression $\Gamma \vdash M : A$, where Γ and M are called the *base* and the *subject* of the sequent, respectively. A base Γ is a set of assignments $x : A$ of types to variables, so that no variable is assigned two different types. The familiar typing rules, which we refrain to repeat, determine the *derivable* sequents, and define a natural deduction system for intuitionitic implicational logic. A λ-term M is *typable* if there is a derivable sequent with M as subject.

System $\Lambda_{\mathtt{v}}$. Plotkin's cbv λ-calculus [11] is here named $\Lambda_{\mathtt{v}}$. The terms of $\Lambda_{\mathtt{v}}$ are the same λ-terms of Λ, but now the system is equipped with a variant of the β-rule, named $\beta_{\mathtt{v}}$: $(\lambda x.M)V \to [V/x]M$. Here, for the function $\lambda x.M$ to be called, the argument is required to be a value. Again, the compatible closure $\to_{\beta_{\mathtt{v}}}$ of $\beta_{\mathtt{v}}$ may be defined by $\mathcal{C}[(\lambda x.M)V] \to_{\beta_{\mathtt{v}}} \mathcal{C}[[V/x]M]$. For the typed version of the system, we employ the same typing system as for Λ.

Several authors [13, 6] have proposed extra reduction principles for $\Lambda_{\mathtt{v}}$. Two of them will be central in the present paper:

$$\begin{array}{lrcl} (\rho_1) & (\lambda x.M)NQ & \to & (\lambda x.MQ)N \\ (\rho_2) & (\lambda y.P)((\lambda x.M)N) & \to & (\lambda x.(\lambda y.P)M)N \end{array}$$

The first is one of Regnier's σ-rules [12]. The author has studied these rules in [3, 4], allowing the second one in a more general form: $P((\lambda x.M)N) \to (\lambda x.PM)N$. A common idea to ρ_1 and ρ_2 is to rearrange the term to reveal (potential) redexes: MQ in the former case, $(\lambda y.P)M$ in the latter case[1]. We let $\rho := \rho_1 \cup \rho_2$.

[1] The name ρ intends to be mnemonic of this action of rearranging to reveal redexes. We find ρ preferable to σ for two reasons: first, σ is a name one wishes to reserve to substitution rules; second, ρ_2 is not one

System ΛJ. The λ-calculus with generalized applications [8] is here named system ΛJ, or the λJ-calculus. The λJ-terms are given by:

$$\text{(Terms) } M,N,P,Q ::= V \mid M(N,x.P) \qquad \text{(Values) } V,W ::= x \mid \lambda x.M$$

The constructor $M(N,x.P)$ is called *generalized application*; in it, "$x.$" is a binder of x, so x is bound in P. Consistent with the generalized application terminology is to call $(N,x.P)$ the *generalized argument* of that application.

ΛJ is equipped with two rules:

$$\begin{array}{lrcl} (\beta) & (\lambda x.M)(N,y.P) & \to & [[N/x]M/y]P \\ (\pi) & M(N,x.P)(N',y.P') & \to & M(N,x.P(N',y.P')) \end{array}$$

The version of π adopted is the same as in [8]. One could have considered an "eager" version, with *contractum* $M(N,x.P@(N',y.P'))$, where operator @ is defined by

$$\begin{array}{rcl} V@(N',y.P') & = & V(N',y.P') \\ M(N,x.P)@(N',y.P') & = & M(N,x.P@(N',y.P')) \end{array}$$

In this version of the rule, the generalized argument $(N',y.P')$ is eagerly pushed in, until a value is found. Operator @ will be used again in Section 5.

The typing system of ΛJ is obtained from the one for Λ by adopting the following rule for typing generalized applications:

$$\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A \quad \Gamma, x : B \vdash P : C}{\Gamma \vdash M(N,x.P) : C}\ GE\supset$$

Such typing system defines a system of natural deduction with generalized elimination rules [16]. Rule π is a "commutative conversion" caused by the repetition of formula C in $GE\supset$.

3 The call-by-value variant

This section is dedicated to the call-by-value variant of ΛJ we introduce, named $\Lambda J_{\mathtt{v}}$. We first motivate the system. In the second part of the section, we define the system.

3.1 Motivation

Consider the β-redex $(\lambda x.M)N$. We will define a new *contractum* for this redex, making use of a syntax where application is generalized. The *contractum* is again a substitution, but one whose definition will express call-by-value - let us denote it by $[N\backslash x]M$.

If $N = V$, no doubt we want to substitute ordinarily, as in Plotkin's $\Lambda_{\mathtt{v}}$, so we put

$$[V\backslash x]M = [V/x]M \tag{1}$$

But if $N = M'N'$, we want to postpone the call of $\lambda x.M$ and evaluate N first. Making use of generalized application, we rewrite the original β-redex as $M'(N',x.M)$. Notice $M'N'$ is actually $M'(N',z.z)$, so we want

$$[M'(N',z.z)\backslash x]M = M'(N',x.M) \tag{2}$$

of Regnier's σ-rules. In [3, 4] these rules were called π_1 and π_2. There is some point in that choice of names, even knowing that π is the name used in [8] for one of the reduction rules of ΛJ - a convention we will maintain here. Still, we found it preferable to abandon π_1 and π_2 and give the rules a fresh and useful name for the present paper.

How to complete the definition of $[N\backslash x]M$, for the case $N = M'(N', z.P')$? Just put

$$[M'(N', z.P')\backslash x]M = M'(N', z.[P'\backslash x]M) \tag{3}$$

Then we get (2) from (3) and (1).

Finally, how about starting off with $(\lambda x.M)(N, y.P)$? As in ΛJ, the *contractum* consists of two substitutions, but both of the new kind we have just defined.

3.2 System ΛJ_{v}

We now define system ΛJ_{v}, or λJ_{v}-calculus. The λJ_{v}-terms are the same as the λJ-terms. Ordinary substitution will be used only in the form $[V/x]M$, with the actual parameter a value V. We introduce **left substitution**, denoted $[N\backslash x]M$, defined by recursion on N as follows:

$$\begin{array}{rcl} [V\backslash x]M & = & [V/x]M \\ [N(Q, y.P)\backslash x]M & = & N(Q, y.[P\backslash x]M) \end{array}$$

ΛJ_{v} has two reduction rules:

$$\begin{array}{lrcl} (\beta_{\mathrm{v}}) & (\lambda x.M)(N, y.P) & \to & [[N\backslash x]M\backslash y]P \\ (\pi) & M(N, x.P)(N', y.P') & \to & M(N, x.P(N', y.P')) \end{array}$$

Rule π is the same as ΛJ, rule β_{v} is new. However, there is a formal similarity with rule β from ΛJ - the only difference is in the substitution operator employed.

The typing system for ΛJ_{v} is the same as the typing system for ΛJ, with left substitution being typed by the same admissible rule that types ordinary substitution. A λJ_{v}-term is typable if it is the subject of some derivable sequent.

A routine result is subject reduction, already known for $\to_\pi$, and which also extends to $\to_{\beta_{\mathrm{v}}}$. Perhaps more important is to recast rule β_{v} as a proof normalization rule in natural deduction with generalized elimination rules - this is done in Fig. 1.

We finish this section with some technical lemmas.

▶ **Lemma 1** (Substitution lemma). *In ΛJ_{v}:*

1. $[V/x][N\backslash y]M = [[V/x]N\backslash y][V/x]M$, *provided* $y \notin V$.
2. $[N\backslash x][N'\backslash y]M = [[N\backslash x]N'\backslash y]M$, *provided* $x \notin M, y \notin N$.

Proof. Routine. ◀

▶ **Lemma 2** (Parallelism). *The following rules are admissible:*

$$\frac{V \to_\pi^* V' \quad M \to_\pi^* M'}{[V/x]M \to_\pi^* [V'/x]M'}\ (i) \qquad \frac{M \to_\pi^* M'}{[N\backslash x]M \to_\pi^* [N\backslash x]M'}\ (ii) \qquad \frac{N \to_\pi^* N' \quad M \to_\pi^* M'}{[N\backslash x]M \to_\pi^* [N'\backslash x]M'}\ (iii)$$

Proof. (i) Known. (ii) Proved by induction on N and uses (i). (iii) Follows easily from the version of (iii) where the first premiss is $N \to_\pi N'$, and the latter is proved by induction on $N \to_\pi N'$, using (i) and (ii). ◀

4 Strong normalization

We define a map from λJ-terms to λ-terms

$$x^\sharp = x \qquad (\lambda x.M)^\sharp = \lambda x.M^\sharp \qquad M(N, x.P)^\sharp = (\lambda x.P^\sharp)(M^\sharp N^\sharp)$$

This map is to be promoted to a homomorphism $\Lambda J_{\mathrm{v}} \to \Lambda_{\mathrm{v}}$. First, the technical lemma:

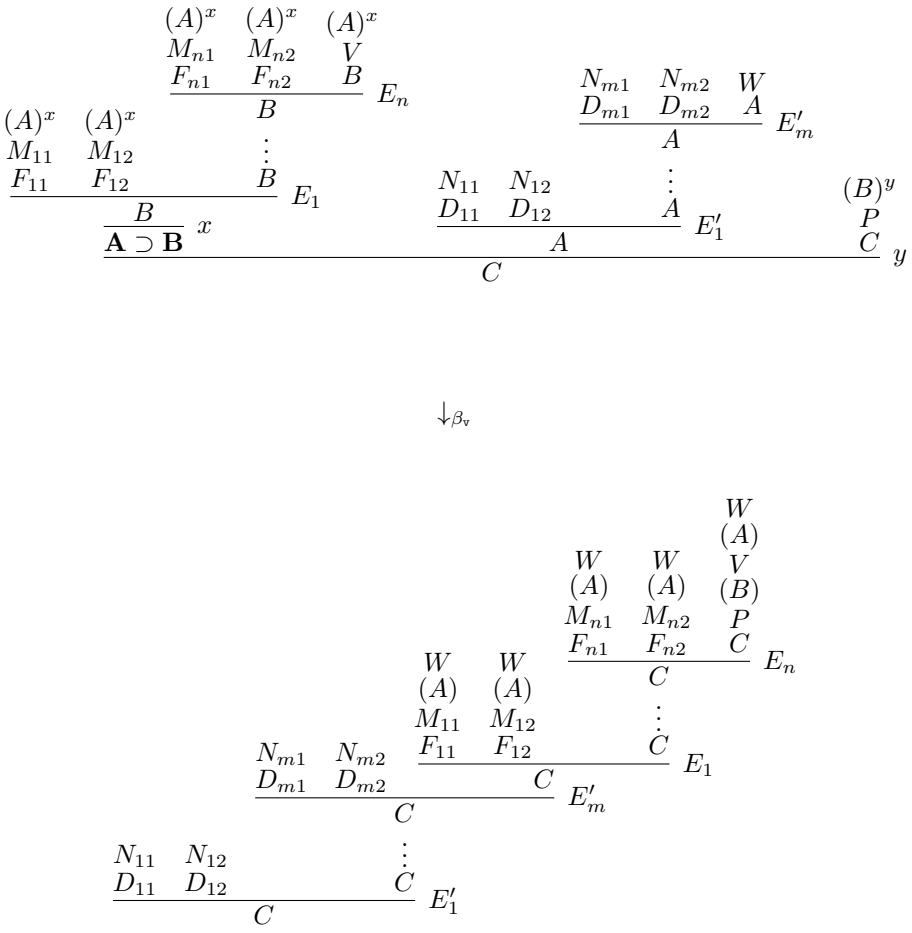

Figure 1 Rule $\beta_{\mathtt{v}}$ as a proof normalization rule in natural deduction with generalized elimination rules. Meta-variables for λJ-terms and values are used to denote derivations. V and W denote derivations whose last inference is not an elimination. The maximal formula is in boldface. The numbers n and m may be 0. D_{11}, D_{12}, D_{m1}, D_{m2}, F_{11}, F_{12}, F_{n1}, F_{n2} are formulas. For each $1 \leq i \leq n$, $F_{i1} = F_{i2} \supset F'_{i2}$, for some formula F'_{i2}. For each $1 \leq i \leq m$, $D_{i1} = D_{i2} \supset D'_{i2}$, for some formula D'_{i2}. Hypothesis cancelation by elimination inferences marked with E_i or E'_i is not displayed.

▶ **Lemma 3.** $(\lambda x.M^\sharp)N^\sharp \to^+_{\beta_{\mathtt{v}}\rho_2} ([N\backslash x]M)^\sharp$.

Proof. By induction on N. The cases $N = y$ and $N = \lambda y.N'$ require the lemma $[V^\sharp/x]M^\sharp = ([V/x]M)^\sharp$, which is proved by a straightforward induction on M. ◀

▶ **Proposition 4** (Strict simulation)**.**

1. *If* $M_1 \to_{\beta_{\mathtt{v}}} M_2$ *in* $\Lambda J_{\mathtt{v}}$ *then* $M_1^\sharp \to^+_{\beta_{\mathtt{v}}\rho_2} M_2^\sharp$ *in* $\Lambda_{\mathtt{v}}$.
2. *If* $M_1 \to_\pi M_2$ *in* $\Lambda J_{\mathtt{v}}$ *then* $M_1^\sharp \to^+_\rho M_2^\sharp$ *in* $\Lambda_{\mathtt{v}}$.

Proof. Both items by induction on $M_1 \to M_2$. The base case of the first item uses Lemma 3 twice. The base case of the second item is proved by a direct calculation. The inductive cases are routine. ◀

▶ **Theorem 5** (SN). *If $M \in \Lambda J_{\mathtt{v}}$ is typable, then M is $\beta_{\mathtt{v}}\pi$-SN.*

Proof. Suppose M is typable. It is easy to see that $M^\sharp$ has to be typable. By strong normalization for Λ, $M^\sharp$ is β-SN. By the main result in [4], $M^\sharp$ is $\beta\rho$-SN, and so is $\beta_{\mathtt{v}}\rho$-SN. Given the strict simulation obtained in Proposition 4 ($M_1 \to_{\beta_{\mathtt{v}}\pi} M_2$ implies $M_1^\sharp \to^+_{\beta_{\mathtt{v}}\rho} M_2^\sharp$), we conclude that M is $\beta_{\mathtt{v}}\pi$-SN. ◀

5 Confluence

In this section we prove that $\to_{\beta_{\mathtt{v}}\pi}$ is confluent. We follow the approach in [8], pointing out where the differences are. In this section, we abbreviate $\to_{\beta_{\mathtt{v}}\pi}$ as $\to$. So $\to^*$ denotes $\to^*_{\beta_{\mathtt{v}}\pi}$.

Given a binary relation $\rightsquigarrow$ on ΛJ-terms and a function f from ΛJ-terms to ΛJ-terms, we say $\rightsquigarrow$ and f satisfy the **triangle property** [15, 8] if $M \rightsquigarrow M'$ implies $M' \rightsquigarrow f(M)$. Hence, every 1-step $\rightsquigarrow$-reduct of M does one $\rightsquigarrow$-step to a common term that depends on M solely; and therefore, $\rightsquigarrow$ satisfies the diamond property.

Given a binary relation $\rightsquigarrow$ on ΛJ-terms, we say $\rightsquigarrow$ is a **$\boldsymbol{\beta_{\mathtt{v}}\pi}$-development**, or just a **development**, if $\to \subseteq \rightsquigarrow \subseteq \to^*$ and $\rightsquigarrow$ and f satisfy the triangle property, for some f (which is then called a **complete $\boldsymbol{\beta_{\mathtt{v}}\pi}$-development**, or just a **development**). Notice that, if there is a development $\rightsquigarrow$, then $\to$ is confluent. Proof: $\rightsquigarrow$ satisfies the diamond property, hence so does $\rightsquigarrow^*$. But $\rightsquigarrow^* = \to^*$ (because $\to \subseteq \rightsquigarrow \subseteq \to^*$). Therefore $\to^*$ satisfies the diamond property, that is, $\to$ is confluent.

For M a ΛJ-term, the π-normal form M^π is defined[2] by recursion on M as follows:

$$\begin{array}{rcl} x^\pi & = & x \\ (\lambda x.M)^\pi & = & \lambda x.M^\pi \\ (M(N, x.P))^\pi & = & M^\pi @ (N^\pi, x.P^\pi) \end{array}$$

▶ **Lemma 6.** *$\to^*_\pi$ and $(\_)^\pi$ satisfy the triangle property.*

Proof. In [8]. ◀

Given that $\to_\pi \subset \to^*_\pi = \to^*_\pi$, we may call $(\_)^\pi$ a complete π-development.

Next we introduce a new pair that will satisfy the triangle property, containing a new binary relation and a complete $\beta_{\mathtt{v}}$-development. The binary relation $\rightrightarrows_{\mathtt{v}}$ on ΛJ is defined in Fig. 2. It is immediate to show that: (i) $\rightrightarrows_{\mathtt{v}}$ is reflexive; (ii) $\to_{\beta_{\mathtt{v}}} \subseteq \rightrightarrows_{\mathtt{v}}$; (iii) $\rightrightarrows_{\mathtt{v}} \subseteq \to^*_{\beta_{\mathtt{v}}}$.

▶ **Lemma 7** (Parallelism). *The following rules are admissible:*

$$\frac{V \rightrightarrows_{\mathtt{v}} V' \quad M \rightrightarrows_{\mathtt{v}} M'}{[V/x]M \rightrightarrows_{\mathtt{v}} [V'/x]M'}\ (i) \qquad \frac{N \rightrightarrows_{\mathtt{v}} N' \quad M \rightrightarrows_{\mathtt{v}} M'}{[N\backslash x]M \rightrightarrows_{\mathtt{v}} [N'\backslash x]M'}\ (ii)$$

Proof. The proof of (i) is by induction on $M \rightrightarrows_{\mathtt{v}} M'$. Case BETA uses item 1 of Lemma 1. The proof of (ii) is by induction on $N \rightrightarrows_{\mathtt{v}} N'$. Cases VAR and ABS use (i). Case BETA uses item 2 of Lemma 1. ◀

[2] The operator @ we are employing in this paper is slightly different from the one employed in [8], but the function M^π is the same.

$$\frac{}{x \rightrightarrows_{\mathtt{v}} x}\ VAR \qquad \frac{M \rightrightarrows_{\mathtt{v}} M'}{\lambda x.M \rightrightarrows_{\mathtt{v}} \lambda x.M'}\ ABS \qquad \frac{M \rightrightarrows_{\mathtt{v}} M' \quad N \rightrightarrows_{\mathtt{v}} N' \quad P \rightrightarrows_{\mathtt{v}} P'}{M(N, x.P) \rightrightarrows_{\mathtt{v}} M'(N', x.P')}\ APL$$

$$\frac{M \rightrightarrows_{\mathtt{v}} M' \quad N \rightrightarrows_{\mathtt{v}} N' \quad P \rightrightarrows_{\mathtt{v}} P'}{(\lambda x.M)(N, y.P) \rightrightarrows_{\mathtt{v}} [[N'\backslash x]M'\backslash y]P'}\ BETA$$

Figure 2 $\beta_{\mathtt{v}}$-development.

For M a ΛJ-term, define $M^{\beta_{\mathtt{v}}}$ by recursion on M as follows:

$$\begin{array}{rcl} x^{\beta_{\mathtt{v}}} & = & x \\ (\lambda x.M)^{\beta_{\mathtt{v}}} & = & \lambda x.M^{\beta_{\mathtt{v}}} \\ (M(N, y.P))^{\beta_{\mathtt{v}}} & = & \begin{cases} [[N^{\beta_{\mathtt{v}}}\backslash x]M_0^{\beta_{\mathtt{v}}}\backslash y]P^{\beta_{\mathtt{v}}} & \text{if } M = \lambda x.M_0 \\ M^{\beta_{\mathtt{v}}}(N^{\beta_{\mathtt{v}}}, y.P^{\beta_{\mathtt{v}}}) & \text{otherwise} \end{cases} \end{array}$$

▶ **Lemma 8.** *$\rightrightarrows_{\mathtt{v}}$ and $(\_)^{\pi}$ satisfy the triangle property.*

Proof. Once we have the parallelism property of $\rightrightarrows_{\mathtt{v}}$ w.r.t. left substitution (item (ii) of Lemma 7), we can repeat the proof in [8]. ◀

Given that $\rightarrow_{\beta_{\mathtt{v}}} \subseteq \rightrightarrows_{\mathtt{v}} \rightarrow^*_{\beta_{\mathtt{v}}}$, we may call $(\_)^{\beta_{\mathtt{v}}}$ a complete $\beta_{\mathtt{v}}$-development.

▶ **Lemma 9** (Commutation)**.** *If $M \rightrightarrows_{\mathtt{v}} N_1$ and $M \rightarrow^*_{\pi} N_2$ then there is P such that $N_1 \rightarrow^*_{\pi} P$ and $N_2 \rightrightarrows_{\mathtt{v}} P$.*

Proof. We can repeat the proof in [8], since $\rightarrow^*_{\pi}$ is also parallel in the sense of Lemma 2. ◀

▶ **Theorem 10.** *$\rightarrow_{\beta_{\mathtt{v}}\pi}$ is confluent.*

Proof. The two triangle properties (Lemmas 6 and 8) and the commutation property (Lemma 9) imply that the relation $\rightarrow^*_{\pi} \circ \rightrightarrows_{\mathtt{v}}$ and the function $(\_)^{\pi} \circ (\_)^{\beta_{\mathtt{v}}}$ have the triangle property (this composition of triangle properties is easily proved - see [8]). Moreover, it is obvious that $\rightarrow_{\beta_{\mathtt{v}}\pi} \subseteq \rightarrow^*_{\pi} \circ \rightrightarrows_{\mathtt{v}} \subseteq \rightarrow^*_{\beta_{\mathtt{v}}\pi}$. This means that $\rightarrow^*_{\pi} \circ \rightrightarrows_{\mathtt{v}}$ is a development (and that $(\_)^{\pi} \circ (\_)^{\beta_{\mathtt{v}}}$ is a complete development). Hence $\rightarrow$ is confluent. ◀

6 Standardization

In this section we prove the mandatory [11] standardization theorem for $\Lambda J_{\mathtt{v}}$. Contrary to many proofs [11, 6], our proof does not handle directly standard reduction sequences - in this we follow [8]. On the other hand, we do not rely on vector notation either; instead we make explicit the contribution of call-by-value evaluation to the standard reduction relation. The definition of cbv evaluation is interesting on its own.

The definition of cbv evaluation is in terms of **cbv evaluation contexts**:

$$\mathcal{E} ::= [\cdot] \,|\, \mathcal{E}(N, x.P)$$

Then **call-by-value evalution**, denoted $\mapsto_{\mathtt{v}}$, is defined as $\mathcal{E}[M] \mapsto_{\mathtt{v}} \mathcal{E}[M]$ where $(M, M') \in \beta_{\mathtt{v}} \cup \pi$ (in other words, $M \rightarrow M'$ is a root $\beta_{\mathtt{v}}\pi$-reduction). The familiar, alternative and equivalent definition is to say that $\mapsto_{\mathtt{v}}$ is inductively defined by $\beta_{\mathtt{v}} \cup \pi$ and the closure rule: $M \rightarrow M'$ implies $M(N, x.P) \rightarrow M'(N, x.P)$.

$$\frac{}{x \Rightarrow x}\ VAR \qquad \frac{M \Rightarrow M'}{\lambda x.M \Rightarrow \lambda x.M'}\ ABS \qquad \frac{M \Rightarrow M' \quad N \Rightarrow N' \quad P \Rightarrow P'}{M(N, x.P) \Rightarrow M'(N', x.P')}\ APL$$

$$\frac{M \mapsto_{\mathtt{v}}^{*} \lambda x.M' \quad [[N\backslash x]M'\backslash y]P \Rightarrow Q}{M(N, y.P) \Rightarrow Q}\ BETA$$

$$\frac{M \mapsto_{\mathtt{v}}^{*} M'(N', x.P') \quad M'(N', x.(P'(N, y.P))) \Rightarrow Q}{M(N, y.P) \Rightarrow Q}\ PI$$

Figure 3 Standard reduction.

So we employ the single closure we would employ if we were defining call-by-name evaluation - the whole difference is in the β-rule. Notice that, due to rule π, evaluation is not a deterministic (univocal) relation.

Standard reduction, denoted $M \Rightarrow M'$, is defined in Fig. 3.

A simple induction shows that $\Rightarrow \subseteq \rightarrow_{\beta_{\mathtt{v}}\pi}^{*}$. Despite its simplicity, this remark is important because its proof builds a $\beta_{\mathtt{v}}\pi$-reduction sequence from M to M', given that $M \Rightarrow M'$. We refer to reduction sequences thus built as **standard reduction sequences**. If $M \Rightarrow M'$ is proved by VAR, ABS or APL, the outer constructor of M is frozen forever. For instance, if $M = \lambda x.M_0$, then $M' = \lambda x.M_0'$ and the reduction sequence given by induction hypothesis will have all of its members prefixed by λx. On the other hand, if $M \Rightarrow M'$ is proved by $BETA$, we prefix the reduction sequence (not its members) given by induction hypothesis by a sequence of cbv evaluation steps, namely the ones leading from $M(N, y.P)$ to $(\lambda x.M')(N, y.P)$ followed by the step $(\lambda x.M')(N, y.P) \mapsto_{\mathtt{v}} [[N\backslash x]M'\backslash y]P$ implicit in rule $BETA$. Similar remarks apply to rule PI. The general description of a standard reduction sequence is this: it contains an initial segment performing cbv evaluation, until one decides to freeze the outer constructor of the last term obtained, and the standard reduction sequence proceeds by reducing in parallel the immediate sub-terms.

The converse of $\Rightarrow \subseteq \rightarrow_{\beta_{\mathtt{v}}\pi}^{*}$ is a kind of completeness of $\Rightarrow$.

▶ **Theorem 11** (Standardization). $\rightarrow_{\beta_{\mathtt{v}}\pi}^{*} \subseteq \Rightarrow$.

Proof. We show successively the closure rules I to VII in Fig. 4. The theorem follows from rule VII (together with rule I). We also need substitutivity of evaluation: If $M \mapsto_{\mathtt{v}}^{*} M'$ then $[V/x]M \mapsto_{\mathtt{v}}^{*} [V'/x]M'$. This is an easy consequence of: If $M \mapsto_{\mathtt{v}} M'$ then $[V/x]M \mapsto_{\mathtt{v}} [V'/x]M'$. The latter is proved by induction on $M \mapsto_{\mathtt{v}} M'$.

Rule I is proved by an easy induction on M. Rule II is proved by induction on $M \mapsto_{\mathtt{v}} M'$. Rule III is an easy consequence of II. Rule IV is proved by induction on $M \Rightarrow M'$ (the case relative to $BETA$ requires substitutivity of evaluation). Rule V is proved by induction on $N \Rightarrow N'$ and uses rule IV. Rule VII is an easy consequence of Rule VI. Rule VI is the rule with most delicate proof. It is proved by induction on $M \Rightarrow M'$. The case relative to APP splits into several cases determined by a reduction step of the form $M'(N', y.P') \rightarrow_{\beta_{\mathtt{v}}\pi} Q$. Two of them are proved with a sub-induction and use rules III and V. ◀

7 Call-by-name and call-by-value

In this section, we show simulations between ΛJ and $\Lambda J_{\mathtt{v}}$. For emphasis, we denote ΛJ by $\Lambda J_{\mathtt{n}}$, and β by $\beta_{\mathtt{n}}$. Both simulation employ the "protecting-by-a-lambda" technique;

$$\frac{}{M \Rightarrow M}\ I$$

$$\frac{M \mapsto_{\mathtt{v}} M' \Rightarrow Q}{M \Rightarrow Q}\ II \qquad \frac{M \mapsto_{\mathtt{v}}^{*} M' \Rightarrow Q}{M \Rightarrow Q}\ III$$

$$\frac{V \Rightarrow V' \quad M \Rightarrow M'}{[V/x]M \Rightarrow [V'/x]M'}\ IV \qquad \frac{N \Rightarrow N' \quad M \Rightarrow M'}{[N\backslash x]M \Rightarrow [N'\backslash x]M'}\ V$$

$$\frac{M \Rightarrow M' \to_{\beta_{\mathtt{v}}\pi} Q}{M \Rightarrow Q}\ VI \qquad \frac{M \Rightarrow M' \to_{\beta_{\mathtt{v}}\pi}^{*} Q}{M \Rightarrow Q}\ VII$$

Figure 4 Additional closure rules.

and, in their typed version, the simulations employ the same type translation, namely the replacement of A by $\top \supset A$ at appropriate places. So, we need neither cps-translations, nor type translations based on the insertion of double negations, in order to translate between the cbn and cbv variants of ΛJ.

We will use I to denote the combinator $\lambda x.x$; and MI will abbreviate $M(I, x.x)$. Also, $\lambda d.M$ will stand for a vacuous abstraction (d is a "dummy" variable, *i.e.* $d \notin M$). We fix some type variable X and put $\top := X \supset X$.

7.1 Simulation of cbn by cbv

For $M \in \Lambda J_{\mathtt{n}}$, M° is defined by recursion on M as follows:

$$\begin{array}{rcl} x^{\circ} & = & xI \\ (\lambda x.M)^{\circ} & = & \lambda x \lambda d.M^{\circ} \\ (M(N, y.P))^{\circ} & = & M^{\circ}(\lambda d.N^{\circ}, y.P^{\circ}) \end{array}$$

This compilation is a variation on the "protecting-by-a-lambda" technique [11], now extended to deal with generalized applications. When translating these, the argument is protected by a vacuous abstraction, to pretend to be a value. Accordingly, variables are applied to a dummy argument. The generality of generalized applications commutes with the translation, but some novelty is observed in the translation of abstractions, where an unexpected, extra, vacuous abstraction shows up.

This surprise has a counterpart in the typed version of this translation, at the level of types. The type A° is defined by recursion on A by:

$$\begin{array}{rcl} X^{\circ} & = & X \\ (A \supset B)^{\circ} & = & \underline{A} \supset \underline{B} \end{array}$$

where $\underline{A} = \top \supset A^{\circ}$. The surprise is that $(A \supset B)^{\circ}$ is not defined as $\underline{A} \supset B^{\circ}$, as one would expect, if this was just the usual "protecting-by-a-lambda", or "thunk-introduction" compilation (see *e.g.* [7, 5]).

As usual, $\underline{\Gamma} = \{(x : \underline{A}) | (x : A) \in \Gamma\}$. It is straightforward to show:

▶ **Proposition 12.** *If* $\Gamma \vdash M : A$ *then* $\underline{\Gamma} \vdash M^{\circ} : A^{\circ}$.

▶ **Lemma 13.** $(\lambda d.M)I \to_{\beta_{\mathtt{v}}} M$.

Proof. By a simple calculation, using the fact $[M\backslash x]x = M$. ◀

▶ **Lemma 14.** $[\lambda d.N^\circ/x]M^\circ \to^*_{\beta_{\mathtt{v}}} ([N/x]M)^\circ$.

Proof. By induction on M. The case $M = x$ uses the previous lemma, and is where the $\beta_{\mathtt{v}}$-steps are generated. ◀

▶ **Theorem 15** (Simulation of $\Lambda J_{\mathtt{n}}$ by $\Lambda J_{\mathtt{v}}$)**.**
1. *If* $M \to_{\beta_{\mathtt{n}}} N$ *in* $\Lambda J_{\mathtt{n}}$ *then* $M^\circ \to^+_{\beta_{\mathtt{v}}} N^\circ$ *in* $\Lambda J_{\mathtt{v}}$.
2. *If* $M \to_\pi N$ *in* $\Lambda J_{\mathtt{n}}$ *then* $M^\circ \to_\pi N^\circ$ *in* $\Lambda J_{\mathtt{v}}$.

Proof. Both items by induction on $M \to N$. The inductive cases are routine. The base case of the first item (case $\beta_{\mathtt{n}}$) uses Lemma 14. The base case of the second item (case π) is a straightforward calculation. ◀

7.2 Simulation of cbv by cbn

In the 1970's [11], the need for cps-translations was justified by the fact that the compilation technique "protecting-by-a-lambda" did not extend to give a simulation of cbv by cbn. Next we show that this is not the case when cbn and cbv are given with generalized applications.

For $V, M \in \Lambda J_{\mathtt{v}}$, $V^\bullet$ and $\overline{M}$ are defined by simultaneous recursion on V and M as follows:

$$\begin{array}{rcl} x^\bullet & = & x \\ (\lambda x.M)^\bullet & = & \lambda x.\overline{M} \\ \overline{V} & = & \lambda d.V^\bullet \\ \overline{M(N, y.P)} & = & \overline{M}(I, m.\overline{N}(I, n.m(n, z.z(I, y.\overline{P})))) \end{array}$$

This time, it is values which are wrapped with vacuous abstractions, and dummy arguments I are used in the translation of application to manipulate the flow of the computation; and even if the translation of applications is reminiscent of cps, the typed version confirms the type structure of this translation is still based on a top level given by $\top \supset A$, and not by a double negation.

The type $A^\bullet$ is defined by recursion on A by:

$$\begin{array}{rcl} X^\bullet & = & X \\ (A \supset B)^\bullet & = & \overline{A} \supset \overline{B} \end{array}$$

where $\overline{A} = \top \supset A^\bullet$. Indeed $A^\bullet = A^\circ$ and $\overline{A} = \underline{A}$.

As usual, $\Gamma^\bullet = \{(x : A^\bullet) | (x : A) \in \Gamma\}$. It is straightforward to show:

▶ **Proposition 16.**
1. *If* $\Gamma \vdash M : A$ *then* $\Gamma^\bullet \vdash \overline{M} : \overline{A}$.
2. *If* $\Gamma \vdash V : A$ *then* $\Gamma^\bullet \vdash V^\bullet : A^\bullet$.

▶ **Lemma 17.**
1. $[V^\bullet/x]\overline{M} = \overline{[V/x]M}$.
2. $\lambda d.[V^\bullet/x]W^\bullet = \overline{[V/x]W}$.

Proof. By simultaneous induction on M and V. ◀

In order to motivate the next two lemmas, observe that the term $I(M, x.P)$ behaves like an explicit substitution, both in $\Lambda J_{\mathtt{n}}$ and $\Lambda J_{\mathtt{v}}$. In the former case, the term reduces to $[M/x]P$; in the latter, it reduces to $[M\backslash x]P$. We would like to have terms that, in $\Lambda J_{\mathtt{n}}$, reduce to $[M\backslash x]P$ and $[[N\backslash x]M\backslash y]P$.

▶ **Lemma 18.** $\overline{M}(I, x.\overline{P}) \to^{+}_{\beta_{\mathtt{n}}\pi} \overline{[M\backslash x]P}$.

Proof. By induction on M. The case $M = V$ uses Lemma 17. ◀

▶ **Lemma 19.** $\overline{N}(I, n.(\lambda x.\overline{M})(n, z.z(I, y.\overline{P}))) \to^{+}_{\beta_{\mathtt{n}}\pi} \overline{[[N\backslash x]M\backslash y]P}$.

Proof. By induction on N. The case $N = V$ uses Lemmas 17 and 18. ◀

▶ **Theorem 20** (Simulation of $\Lambda J_{\mathtt{v}}$ by $\Lambda J_{\mathtt{n}}$)**.**
1. *If* $M \to_{\beta_{\mathtt{v}}} N$ *in* $\Lambda J_{\mathtt{v}}$ *then* $\overline{M} \to^{+}_{\beta_{\mathtt{n}}\pi} \overline{N}$ *in* $\Lambda J_{\mathtt{n}}$.
2. *If* $M \to_{\pi} N$ *in* $\Lambda J_{\mathtt{v}}$ *then* $\overline{M} \to_{\pi} \overline{N}$ *in* $\Lambda J_{\mathtt{n}}$.

Proof. Both items by indunction on $M \to N$. The inductive cases are routine. The base case of the first item (case $\beta_{\mathtt{v}}$) uses Lemma 19. The base case of the second item (case π) is a straightforward calculation. ◀

8 Final remarks

We identified a call-by-value variant $\Lambda J_{\mathtt{v}}$ of system ΛJ, sharing the set of terms with the original, cbn variant, and only differing in the definition of substitution. We established the main rewriting-theoretic properties (strong normalization and confluence), and proved the standardization theorem in a way that makes evident the contribution of call-by-value evaluation for standard reduction. Finally, we proved that the cbn and cbv variant simulate each other, not via cps-translations, but rather via the technique of "protecting-by-a-lambda"[11], or "thunk-introduction"[7], which is here shown for the first time to extend to a simulation of cbv by cbn.

In [5] one sees the simulation of the cbn, ordinary λ-calculus and of Plotkin's cbv λ-calculus into a common modal language, via modal embeddings: cps-translations are dispensed with, because use can be made of the extra facilities of the modal target. But here, no extension of the logic is required, we never leave intuitionistic implicational logic. Instead, use is made of the structural extension provided by generalized applications.

Our goal was to define the cbv variant of ΛJ and the cbv variant of natural deduction with generalized elimination rules: we believe this was not attempted before, we made a proposal and studied it. On the other hand, the study of cbv λ-calculi is an active field of research, with new calculi being proposed for decades ([11, 9, 13, 14, 2, 1, 6]). Although this was not our primary goal, we believe we made a contribution to this line of work, since $\Lambda J_{\mathtt{v}}$ seems to have a singular place among the panoply of systems in the literature, for various reasons: first, it is strongly anchored in proof theory; second: it is extremely simple; third, it exploits the original syntactic idea of fusing into a single constructor (generalized application) ordinary application with let-expressions.

Finally, there may be another reason for studying $\Lambda J_{\mathtt{v}}$ further: contrary to, say, Plotkin's cbv λ-calculus, β-redexes in $\Lambda J_{\mathtt{v}}$ always reduce, never get stuck. Now this may turn the system suitable for "open" call-by-value [1] - but that remains future work.

References

1 B. Accattoli and G. Guerrieri. Open Call-by-Value. In *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016.

2 R. Dyckhoff and S. Lengrand. Call-by-value lambda calculus and LJQ. *Journal of Logic and Computation*, 17:1109–1134, 2007.

3 J. Espírito Santo. Delayed substitutions. In Franz Baader, editor, *Proceedings of RTA'07*, volume 4533 of *Lecture Notes in Computer Science*, pages 169–183. Springer-Verlag, 2007.

4 J. Espírito Santo. A note on preservation of strong normalisation in the λ-calculus. *Theoretical Computer Science*, 412(11):1027–1032, 2011.

5 J. Espírito Santo, L. Pinto, and T. Uustalu. Modal Embeddings and Calling Paradigms. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany.*, volume 131 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

6 G. Guerrieri, L. Paolini, and S. Ronchi Della Rocca. Standardization and Conservativity of a Refined Call-by-Value lambda-Calculus. *Logical Methods in Computer Science*, 13(4), 2017.

7 J. Hatcliff and O. Danvy. Thunks and the λ-Calculus (Extended Version). Technical Report BRICS RS-97-7, DIKU, 1997.

8 F. Joachimski and R. Matthes. Standardization and Confluence for a Lambda Calculus with Generalized Applications. In *Proceedings of RTA 2000*, volume 1833 of *LNCS*, pages 141–155. Springer, 2000.

9 E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.

10 S. Negri and J. von Plato. *Structural Proof Theory.* Cambridge University Press, 2001.

11 G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

12 L. Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126(2):281–292, 1994.

13 A. Sabry and M. Felleisen. Reasoning about programms in continuation-passing-style. *LISP and Symbolic Computation*, 6(3/4):289–360, 1993.

14 A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Trans. on Programming Languages and Systems*, 19(6):916–941, 1997.

15 M. Takahashi. Parallel reductions in λ-calculus. *Information & Computation*, 118:120–127, 1995.

16 J. von Plato. Natural deduction with general elimination rules. *Annals of Mathematical Logic*, 40(7):541–567, 2001.