A Deeper Study of $\lambda!$-Calculus Simulations

Victor Arrial

Université Paris Cité, CNRS, IRIF, France

Abstract

In this paper we study properties of the encodings of CBN and CBV into a CBPV like language

1 Introduction

Bang Calculus. P.B. Levy introduced Call-by-Push-Value (CBPV) [13] as a *subsuming* language, so that different evaluation strategies of the λ-calculus can be captured in a uniform framework by the simple use of two primitives: *thunk* (to pause a computation) and *force* (to resume a computation). This mechanism is powerful enough to encode, in particular, Call-by-Name (CBN) and Call-by-Value [14] (CBV). The original CBPV has been introduced in a simply typed framework, but the underlying (untyped) syntax and operational semantics –the ones we are interested in here– already provide a powerful untyped subsuming mechanism. Despite that, CBN and CBV have always been studied notably by developing different techniques for one and the other. Some rare exceptions are [5, 8, 12, 3], where some particular property for CBN/CBV (e.g. quantitative typing, factorization, tight typing, inhabitation) is derived from the corresponding property for a language that is a restriction of CBPV, via a suitable CBN/CBV encoding. Such a language can be the *bang calculus* [7, 10, 11] (in turn inspired by Ehrhard [6], combining ideas from Levy's CBPV [13] and Girard's linear logic [9]), or its variant the $\lambda!$-*calculus* [5, 12] where reduction rules act *at a distance*.

Normal Form, Simulation and Reverse Simulation. In this work, we are interested in the preservation of the reduction relation by the encoding. On one hand and from a static point of view, one may consider the *preservation of normal forms*: t is a normal form if and only if t^e is a normal form, where t^e is the e-encoding of t. On the second hand and from a more dynamical point of view, two properties can naturally be considered. *Simulation*: reduction steps are transported into the subsuming paradigm (i.e. if $t \to u$ in the subsumed language then $t^e \to u^e$ in the subsuming language). *Reverse simulation*: reduction steps are recovered from the subsuming paradigm (i.e. if $t^e \to u^e$ in the subsuming language then $t \to u$ in the subsumed language).

Some of these properties have been studied for CBPV [13], bang calculus [10] and $\lambda!$-calculus [5] but a full analysis is still missing. In this work, we look into the normal form preservation, simulation and reverse simulation for the $\lambda!$-calculus. In particular, taking into account distance is not a trivial matter in that it requires modifications of the call-by-value encoding.

2 The $\lambda!$-Calculus

We now briefly introduce the $\lambda!$-calculus [5]. Given a countably infinite set $\mathcal{X}$ of variables $x, y, z, \ldots$, the set $\Lambda_!$ of terms is given by the following inductive definition:

$$\textbf{(Terms)} \qquad t, u \quad ::= \quad x \in \mathcal{X} \mid \lambda x.t \mid tu \mid t[x \backslash u] \mid !t \mid \mathtt{der}(t)$$

$$
\begin{array}{rclrcl}
x^{\mathtt{cbn}} & = & x & x^{\mathtt{cbv}} & = & !x \\
(\lambda x.t)^{\mathtt{cbn}} & = & \lambda x.t^{\mathtt{cbn}} & (\lambda x.t)^{\mathtt{cbv}} & = & !\lambda x.t^{\mathtt{cbv}} \\
(tu)^{\mathtt{cbn}} & = & t^{\mathtt{cbn}}!u^{\mathtt{cbn}} & (tu)^{\mathtt{cbv}} & = & \begin{cases} \mathtt{L}\langle s\rangle\, u^{\mathtt{cbv}} & \text{if } t^{\mathtt{cbv}} = \mathtt{L}\langle !s\rangle \\ \mathtt{der}(t^{\mathtt{cbv}})u^{\mathtt{cbv}} & \text{otherwise} \end{cases} \\
(t[x\backslash u])^{\mathtt{cbn}} & = & t^{\mathtt{cbn}}[x\backslash !u^{\mathtt{cbn}}] & (t[x\backslash u])^{\mathtt{cbv}} & = & t^{\mathtt{cbv}}[x\backslash u^{\mathtt{cbv}}]
\end{array}
$$

Figure 1: CbN and CbV Embeddings for $\lambda!$-Calculus [5]

The set $\Lambda_!$ includes λ-terms (variables x, abstractions $\lambda x.t$ and applications tu) as well as three new constructors: a closure $t[x\backslash u]$ representing a pending explicit substitution $[x\backslash u]$ on a term t, a bang $!t$ to freeze the execution of t, and a dereliction $\mathtt{der}(t)$ to fire again the frozen term t. The usual notion of α-conversion [4] is extended to the whole set $\Lambda_!$, and terms are identified up to α-conversion. We denote by $t\{x := v\}$ the usual (capture avoiding) meta-level substitution of the term u for all free occurrences of the variable x in the term t.

The sets of list contexts $\mathtt{L}$, surface contexts $\mathtt{S}$ and full contexts $\mathtt{F}$, which can be seen as terms containing exactly one hole $\diamond$, are inductively defined as follows:

$$
\begin{array}{rcl}
\textbf{(List)}\ \mathtt{L} & ::= & \diamond \mid \mathtt{L}[x\backslash t] \\
\textbf{(Surface)}\ \mathtt{S} & ::= & \diamond \mid \lambda x.\mathtt{S} \mid \mathtt{S}\,t \mid t\,\mathtt{S} \mid \mathtt{S}[x\backslash t] \mid t[x\backslash \mathtt{S}] \mid \mathtt{der}(\mathtt{S}) \\
\textbf{(Full)}\ \mathtt{F} & ::= & \diamond \mid \lambda x.\mathtt{F} \mid \mathtt{F}\,t \mid t\,\mathtt{F} \mid \mathtt{F}[x\backslash t] \mid t[x\backslash \mathtt{F}] \mid \mathtt{der}(\mathtt{F}) \mid !\mathtt{F}
\end{array}
$$

The hole can occur everywhere in $\mathtt{F}$, while in $\mathtt{S}$ it cannot occur under a !. We write $\mathtt{L}\langle t\rangle$ for the term obtained by replacing the hole in $\mathtt{L}$ with the term t and similarly for $\mathtt{S}$ and $\mathtt{F}$.

The following three rewriting rules are the base components of our reduction relations:

$$
\mathtt{L}\langle \lambda x.t\rangle\, u \ \mapsto_{\mathtt{dB}}\ \mathtt{L}\langle t[x\backslash u]\rangle \qquad t[x\backslash \mathtt{L}\langle !u\rangle] \ \mapsto_{\mathtt{s!}}\ \mathtt{L}\langle t\{x := u\}\rangle \qquad \mathtt{der}(\mathtt{L}\langle !t\rangle) \ \mapsto_{\mathtt{d!}}\ \mathtt{L}\langle t\rangle
$$

Rule $\mathtt{dB}$ (resp. $\mathtt{s!}$) is assumed to be capture free, so no free variable of u (resp. t) is captured by the context $\mathtt{L}$. The rule $\mathtt{dB}$ fires a standard β-redex and generates an explicit substitution. The rule $\mathtt{s!}$ fires an explicit substitution provided that its argument is a bang. The rule $\mathtt{d!}$ defrosts a frozen term. In all of these rewrite rules, the reduction acts at a distance [1]: the main constructors involved in the rule can be separated by a finite –possibly empty– list $\mathtt{L}$ of explicit substitutions. This mechanism unblocks redexes that otherwise would be stuck, e.g. $(\lambda x.x)[y\backslash w]!z \mapsto_{\mathtt{dB}} x[x\backslash !z][y\backslash w]$ fires a β-redex by taking $\mathtt{L} = \diamond[y\backslash w]$ as the list context in between the function $\lambda x.x$ and the argument $!z$.

The surface reduction relation $\rightarrow_{\mathtt{S}}$ is the surface closure of any of the three rewrite rules $\mathtt{dB}$, $\mathtt{s!}$ and $\mathtt{d!}$, i.e. $\rightarrow_{\mathtt{S}}$ only fires redexes in surface contexts, and not under bang. Similarly, the full reduction relation $\rightarrow_{\mathtt{F}}$ is the full closure of any of the rewrite rules, so that $\rightarrow_{\mathtt{F}}$ reduces under full contexts and thus the bang loses its freezing behavior. For example,

$$
(\lambda x.!\mathtt{der}(!x))!y \ \rightarrow_{\mathtt{S}}\ (!\mathtt{der}(!x))[x\backslash !y] \ \rightarrow_{\mathtt{S}}\ !\mathtt{der}(!y) \ \rightarrow_{\mathtt{F}}\ !y
$$

Note that the first two steps are also $\rightarrow_{\mathtt{F}}$-steps, while the last step is not an $\rightarrow_{\mathtt{S}}$-step. More generally, we have $\rightarrow_{\mathtt{S}} \subseteq \rightarrow_{\mathtt{F}}$. Moreover, we denote by $\twoheadrightarrow_{\mathtt{S}}$ (resp. $\twoheadrightarrow_{\mathtt{F}}$) the reflexive and transitive closure of $\rightarrow_{\mathtt{S}}$ (resp. $\rightarrow_{\mathtt{F}}$).

3 Call-by-Name – $\lambda_{\mathtt{n}}$

We now briefly introduce the (call-by-name) $\lambda_{\mathtt{n}}$-calculus. Given a countably infinite set $\mathcal{X}$ of variables $x, y, z, ...$, the set $\Lambda_{\mathtt{n}}$ of terms is given by the following inductive definition:

$$\textbf{(Terms)} \qquad t, u \quad ::= \quad x \in \mathcal{X} \mid \lambda x.t \mid tu \mid t[x \backslash u]$$

Terms are identified up the the usual notion of α-conversion [4] and we denote by $t\{x := v\}$ the expected (capture avoiding) meta-level substitution. The sets of list contexts $\mathtt{L}$, surface contexts $\mathtt{N}$ and full contexts $\mathtt{C}$, are inductively defined as follows:

$$\begin{array}{rlll} \textbf{(List)} & \mathtt{L} & ::= & \diamond \mid \mathtt{L}[x \backslash t] \\ \textbf{(Surface)} & \mathtt{N} & ::= & \diamond \mid \lambda x.\mathtt{N} \mid \mathtt{N}\,t \mid \mathtt{N}[x \backslash t] \\ \textbf{(Full)} & \mathtt{C} & ::= & \diamond \mid \lambda x.\mathtt{C} \mid \mathtt{C}\,t \mid t\,\mathtt{C} \mid \mathtt{C}[x \backslash t] \mid t[x \backslash \mathtt{C}] \end{array}$$

The hole can occur everywhere in $\mathtt{C}$, while in $\mathtt{N}$ it cannot occur in the argument of an application nor an explicit substitution.

The following rewriting rules are the base components of our reduction relations:

$$\mathtt{L}\langle \lambda x.t \rangle\, u \quad \mapsto_{\mathtt{dB}} \quad \mathtt{L}\langle t[x \backslash u] \rangle \qquad\qquad t[x \backslash u] \quad \mapsto_{\mathtt{s}} \quad t\{x := u\}$$

The surface reduction relation $\rightarrow_{\mathtt{N}}$ (resp. full reduction relation $\rightarrow_{\mathtt{C}}$) is defined as the surface closure $\mathtt{N}$ (resp. full closure $\mathtt{C}$) of the rules $\mathtt{dB}$ and $\mathtt{s}$ presented above. Moreover, we denote by $\twoheadrightarrow_{\mathtt{N}}$ (resp. $\twoheadrightarrow_{\mathtt{C}}$) the reflexive transitive closure of the surface reduction $\rightarrow_{\mathtt{N}}$ (resp. full reduction $\rightarrow_{\mathtt{C}}$). Finally, a term $t \in \Lambda_{\mathtt{n}}$ is said in to be a surface (resp. full) normal form if there is no u such that $t \rightarrow_{\mathtt{N}} u$ (resp. $t \rightarrow_{\mathtt{C}} u$).

The embedding $\cdot^{\mathtt{cbn}}$ presented in (Fig. 1) has been shown [5] to preserve surface normal forms. However, this property can be strengthen to also include full normal forms:

Theorem 3.1 (Normal Forms Preservation). *Let $t \in \Lambda_{\mathtt{n}}$, then t is a surface (resp. full) normal form if and only if $t^{\mathtt{cbn}}$ is a surface (resp. full) normal form.*

Rather than just looking at static properties, we are actually interested in dynamic ones. In particular, surface reduction is well transported into the bang calculus (simulation property in [5]). Interestingly, we show that it also holds for the full reduction. Moreover, we show that reverse simulation is also verified for both surface and full reductions.

Theorem 3.2 (Surface/Full Simulation and Reverse Simulation). *Let $t, u \in \Lambda_{\mathtt{n}}$, then:*

- *$t \rightarrow_{\mathtt{N}} u$ (resp. $t \twoheadrightarrow_{\mathtt{N}} u$) if and only if $t^{\mathtt{cbn}} \rightarrow_{\mathtt{S}} u^{\mathtt{cbn}}$ (resp. $t^{\mathtt{cbn}} \twoheadrightarrow_{\mathtt{S}} u^{\mathtt{cbn}}$).*
- *$t \rightarrow_{\mathtt{C}} u$ (resp. $t \twoheadrightarrow_{\mathtt{C}} u$) if and only if $t^{\mathtt{cbn}} \rightarrow_{\mathtt{F}} u^{\mathtt{cbn}}$ (resp. $t^{\mathtt{cbn}} \twoheadrightarrow_{\mathtt{F}} u^{\mathtt{cbn}}$).*

Moreover, the number of $\mathtt{dB}$ (resp. $\mathtt{s}$) steps exactly matches the number of $\mathtt{dB}$ (resp. $\mathtt{s!}$) steps.

4 Call-by-Value – $\lambda_{\mathtt{vsub}}$

We now briefly introduce the (distant call-by-value) $\lambda_{\mathtt{vsub}}$-calculus [2]. The set $\Lambda_{\mathtt{v}}$ of terms is given by the following inductive definitions:

$$\begin{array}{rrll} \textbf{(Terms)} & t, u & ::= & v \mid tu \mid t[x \backslash u] \\ \textbf{(Values)} & v & ::= & x \in \mathcal{X} \mid \lambda x.t \end{array}$$

$$\begin{tikzcd}
\text{In } \Lambda_{\mathtt{n}}: & (\lambda x.((\lambda y.y)z))z \arrow[r, phantom, "\not\to_{\mathtt{V}}"] \arrow[d, "\cdot^{\mathtt{cbv}}"] & (\lambda x.(y[y\backslash z]))z \arrow[d, "\cdot^{\mathtt{cbv}}"] \\
\text{In } \Lambda_{!}: & (\lambda x.((\lambda y.!y)\,(!z)))\,(!z) \arrow[r, "\mathtt{S}"] & (\lambda x.((!y)[y\backslash !z]))\,(!z)
\end{tikzcd}$$

Figure 2: A Counterexample of the CbV Reverse Simulation Property

$$\begin{array}{rcll}
x^{\mathtt{CBV}} & = & !x & \\
(\lambda x.t)^{\mathtt{CBV}} & = & !\lambda x.!t^{\mathtt{CBV}} & \\
(tu)^{\mathtt{CBV}} & = & \begin{cases} \mathtt{der}(\mathtt{L}\langle s\rangle\, u^{\mathtt{CBV}}) & \text{if } t^{\mathtt{CBV}} = \mathtt{L}\langle !s\rangle \\ \mathtt{der}(\mathtt{der}(t^{\mathtt{CBV}})u^{\mathtt{CBV}}) & \text{otherwise} \end{cases} & \\
(t[x\backslash u])^{\mathtt{CBV}} & = & t^{\mathtt{CBV}}[x\backslash u^{\mathtt{CBV}}] &
\end{array}$$

Figure 3: A New Call-by-Value Embedding for Bang-calculus

Again, we consider the set of terms up to the usual α-conversion [4] and we denote by $t\{x := v\}$ the expected (capture avoiding) meta-level substitution. The sets of list contexts $\mathtt{L}$, surface contexts $\mathtt{V}$ and full contexts $\mathtt{C}$, are inductively defined as follows:

$$\begin{array}{rrcl}
\textbf{(List)} & \mathtt{L} & ::= & \diamond \mid \mathtt{L}[x\backslash t] \\
\textbf{(Surface)} & \mathtt{V} & ::= & \diamond \mid \mathtt{V}\,t \mid t\,\mathtt{V} \mid \mathtt{V}[x\backslash t] \mid t[x\backslash \mathtt{V}] \\
\textbf{(Full)} & \mathtt{C} & ::= & \diamond \mid \lambda x.\mathtt{C} \mid \mathtt{C}\,t \mid t\,\mathtt{C} \mid \mathtt{C}[x\backslash t] \mid t[x\backslash \mathtt{C}]
\end{array}$$

In particular, the hole can occur everywhere in $\mathtt{C}$, while in $\mathtt{V}$ it cannot occur under an abstraction. The following rewriting rules are the base components of our reduction relations.

$$\mathtt{L}\langle \lambda x.t\rangle\, u \quad \mapsto_{\mathtt{dB}} \quad \mathtt{L}\langle t[x\backslash u]\rangle \qquad\qquad t[x\backslash \mathtt{L}\langle v\rangle] \quad \mapsto_{\mathtt{sV}} \quad \mathtt{L}\langle t\{x := v\}\rangle$$

In both these rules the reduction acts at a distance in order to deal with stuck redexes since this phenomenon also appear in call-by-value. The surface reduction relation $\to_{\mathtt{V}}$ (resp. full reduction relation $\to_{\mathtt{C}}$) is defined as the surface closure $\mathtt{V}$ (resp. full closure $\mathtt{C}$) of the rules $\mathtt{dB}$ and $\mathtt{sV}$ presented above. We denote by $\twoheadrightarrow_{\mathtt{V}}$ (resp. $\twoheadrightarrow_{\mathtt{C}}$) the reflexive transitive closure of the surface reduction $\to_{\mathtt{V}}$ (resp. full reduction $\to_{\mathtt{C}}$) and finally, a term $t \in \Lambda_{\mathtt{v}}$ is said in to be a surface (resp. full) normal form if there is no u such that $t \to_{\mathtt{V}} u$ (resp. $t \to_{\mathtt{C}} u$).

The embedding $\cdot^{\mathtt{cbv}}$ presented in Fig. 1 preserves surface normal forms and satisfies the simulation property for the surface reduction [5]. However, reverse simulation fails for the surface reduction. A counter example can be found in Fig 2.

We introduce in Fig. 3 a new call-by-value embedding $\cdot^{\mathtt{CBV}}$ solving the issue. The main difference can be found in the abstraction case where two ! are used instead of one. The application is then adapted by placing an additional dereliction on the outside. This dereliction gets rid of the bang on the body of the abstraction once the $\mathtt{dB}$ redex has been fired. This restores access to the body of the previously existing abstraction, matching the phenomenon at play in distance.

This new embedding has the same good static property of preserving the surface normal forms and is additionally shown to preserve full normal forms.

Theorem 4.1 (Normal Forms Preservation)**.** *Let $t \in \Lambda_{\mathtt{v}}$, then t is a surface (resp. full) normal form if and only if $t^{\mathtt{CBV}}$ is a surface (resp. full) normal form.*

Moreover and as intended, this new encoding is satisfying both simulation and reverse simulation properties for both surface and full reductions:

Theorem 4.2 (Surface/Full Simulation and Reverse Simulation)**.** *Let* $t, u \in \Lambda_{\mathtt{vsub}}$*, then:*

- $t \rightarrow_{\mathtt{V}} u$ *(resp.* $t \twoheadrightarrow_{\mathtt{V}} u$*) if and only if* $t^{\mathtt{CBV}} \twoheadrightarrow_{\mathtt{S}} u^{\mathtt{CBV}}$.
- $t \rightarrow_{\mathtt{C}} u$ *(resp.* $t \twoheadrightarrow_{\mathtt{C}} u$*) if and only if* $t^{\mathtt{CBV}} \twoheadrightarrow_{\mathtt{F}} u^{\mathtt{CBV}}$.

where the number of $\mathtt{dB}$ *(resp.* $\mathtt{s}$*) steps exactly matches the number of* $\mathtt{dB}$ *(resp.* $\mathtt{s!}$*) steps.*

References

[1] Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *Proceedings of 24th EACSL Conference on Computer Science Logic*, volume 6247 of *LNCS*, pages 381–395. Springer, August 2010.

[2] Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In Tom Schrijvers and Peter Thiemann, editors, *FLOPS*, volume 7294 of *LNCS*, pages 4–16. Springer, 2012.

[3] Victor Arrial, Giulio Guerrieri, and Delia Kesner. Quantitative inhabitation for different lambda calculi in a unifying framework. *Proc. ACM Program. Lang.*, 7(POPL):1483–1513, 2023.

[4] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundation of mathematics*. North-Holland, Amsterdam, revised edition, 1984.

[5] Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. In Keisuke Nakano and Konstantinos Sagonas, editors, *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*, volume 12073 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2020.

[6] Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In Peter Thiemann, editor, *ESOP 2016*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016.

[7] Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In James Cheney and Germán Vidal, editors, *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016*, pages 174–187. ACM, 2016.

[8] Claudia Faggian and Giulio Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In Stefan Kiefer and Christine Tasson, editors, *FOSSACS 2021*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021.

[9] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[10] Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity-TLLA@FLoC 2018*, volume 292 of *EPTCS*, pages 15–30, 2018.

[11] Giulio Guerrieri and Federico Olimpieri. Categorifying non-idempotent intersection types. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 25:1–25:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[12] Delia Kesner and Andrés Viso. Encoding tight typing in a unified framework. *CoRR*, abs/2105.00564, 2021.

[13] Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, pages 228–243, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[14] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.