Call-by-Value Typing Revisited, for Free ?

Victor Arrial

Université Paris Cité, CNRS, IRIF, France

Abstract. In this paper, we introduce an alternative quantitative typing system $\mathcal{V}'$ for dCBV.

1 Introduction

Bang Calculus. P.B. Levy introduced Call-by-Push-Value (CBPV) [19] as a *subsuming* language to encompass various evaluation strategies of the λ-calculus through two simple 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 [20] (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 robust subsuming mechanism. Drawing inspiration from it and Girard's Linear Logic (LL) [15], Ehrhard and Guerrieri [12] introduced an (untyped) restriction of CBPV, named BANG-calculus, already capable of subsuming both CBN and CBV. It is obtained by enriching the λ-calculus with two modalities ! and its dual der. The modality ! actually plays a twofold role: it freezes the evaluation of subterms (called *thunk* in CBPV), and it marks what can be duplicated or erased during evaluation (*i.e.* copied an arbitrary number of times, including zero). The modality der annihilates the effect of !, effectively restoring computation and eliminating duplicability. Embedding CBN or CBV into the BANG-calculus simply consists in decorating λ-terms with ! and der, thereby forcing one model of computation or the other one. Thanks to these elementary modalities and embeddings, the Bang Calculus eases the identification of shared behaviors and properties, encompassing both syntactic and semantic aspects.

Static and dynamic. The literature has shown that some *static* properties (*i.e.* centered on terms) of CBN and CBV, including normal forms [18], quantitative typing [9], tight typing [18, 10], inhabitation [5], and denotational semantics [16], can be inferred from their corresponding counterparts in the Bang Calculus by exploiting suitable CBN and CBV encodings. However, retrieving *dynamic* properties (*i.e.* centered on reduction) from the Bang Calculus into CBN or CBV turned out to be a more intricate task, especially in their *adequate* (distant) variant [16, 9, 13, 10]. Indeed, it is easy to obtain *simulation* (a CBN or CBV reduction sequence is always embedded into a BANG reduction sequence), *but* the converse, known as *reverse simulation*, fails [6, 10]: a BANG reduction sequence from a term in the image of the CBN or CBV embedding may not correspond to a valid reduction sequence in CBN or CBV. To recover this property, an alternative CBV encoding was introduced in [6] and allows to project *for free* of the diamond property, confluence and factorization directly from BANG to CBN and CBV. Yet, all use cases of this new embedding have been limited to syntactical properties.

Contributions. This paper concentrates on the preservation of typed properties through the newly introduced CBV embedding [6]. We propose an alternative quantitative typing system for the distant CBV λ-calculus and prove that it is preserved by the embedding. Additionally, we prove *for free* the subject reduction and expansion property, along with soundness and completeness of this new typing system by a simple projection from the distant BANG-calculus and its quantitative typing system.

2 The (Distant) Bang Calculus

Syntax. We introduce the term syntax of dBang [9]. Given a countably infinite set $\mathcal{X}$ of variables $x, y, z, \ldots$, the set $\Lambda_!$ of terms is defined inductively as follows:

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

Abstractions $\lambda x.t$ and closures $t[x\backslash u]$ bind the variable x in their body t. **Free** and **bound variables** are defined as expected. The usual notion of α-conversion [8] is extended to the whole set $\Lambda_!$, and terms are identified up to α-conversion. Finally, we denote by $t\{x\backslash u\}$ the usual (capture avoiding) meta-level substitution of u for all free occurrences of x in t.

Full contexts (F), **surface contexts** (S) and **list contexts** (L), which can be seen as terms with exactly one **hole** $\diamond$, are inductively defined by:

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

We write $\mathtt{F}\langle t\rangle$ for the term obtained by replacing the hole in F with the term t (possibly capturing the free variables of t).

Operational Semantics. The following **rewrite rules** are the base components of our reductions.

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

Rule dB (resp. s!) is assumed to be capture-free, so no free variable of u (resp. t) is captured by the context L. In all of these rewrite rules, the reduction acts *at a distance* [1, 2]: the main constructors involved in the rule can be separated by a finite —possibly empty— list L of ES.

The **surface** (resp. **full**) **reduction** $\rightarrow_{\mathrm{S}}$ (resp. $\rightarrow_{\mathrm{F}}$) is the defined as the union of the closure of the relations $\{\mathtt{dB}, \mathtt{s!}, \mathtt{d!}\}$ by surface (resp. full) contexts. For example, $(\lambda x.!\mathtt{der}(!x))!y \ \rightarrow_{\mathrm{S}} \ (!\mathtt{der}(!x))[x\backslash !y] \ \rightarrow_{\mathrm{S}} \ !(\mathtt{der}(!y)) \ \rightarrow_{\mathrm{F}} \ !y$. Finally, we denote by $\rightarrow_{\mathrm{F}}^*$ (resp. $\rightarrow_{\mathrm{S}}^*$) the reflexive transitive closure of $\rightarrow_{\mathrm{F}}$ (resp. $\rightarrow_{\mathrm{S}}$).

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (var) \qquad \frac{\Gamma \vdash t : \mathcal{M} \Rightarrow \sigma \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t\,u : \sigma}\ (app) \qquad \frac{(\Gamma_i \vdash t : \sigma_i)_{i \in I}}{+_{i\in I}\Gamma_i \vdash\, !t : [\sigma_i]_{i \in I}}\ (bg)$$

$$\frac{\Gamma, x : \mathcal{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \mathcal{M} \Rightarrow \sigma}\ (abs) \qquad \frac{\Gamma, x : \mathcal{M} \vdash t : \sigma \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t[x \backslash u] : \sigma}\ (es) \qquad \frac{\Gamma \vdash t : [\sigma]}{\Gamma \vdash \mathtt{der}(t) : \sigma}\ (der)$$

Fig. 1: Type System $\mathcal{B}$ for the dBang-calculus.

Clashes. As a matter of fact, some ill-formed terms are not redexes but neither represent a desired computation result. They are called **clashes** and have one of the following forms:

$$\mathtt{L}\langle !t\rangle\, u \qquad t[x\backslash \mathtt{L}\langle \lambda x.u\rangle] \qquad \mathtt{der}(\mathtt{L}\langle \lambda x.t\rangle) \qquad t(\mathtt{L}\langle \lambda x.u\rangle) \text{ if } t \neq \mathtt{L}'\langle \lambda y.s\rangle$$

This *static* notion of clash is lifted to a *dynamic* level. A term t is a **surface clash-free** if it does not surface reduce to a term with a clash in surface position. This notion is stable under reduction.

Quantitative Type System. We present the quantitative typing system $\mathcal{B}$ [9], based on [14, 11]. It contains functional and intersection types. Here, intersections are associative, commutative but *not idempotent*, thus an intersection type is represented by a (possibly empty) *finite multiset* $[\sigma_i]_{i\in I}$. Formally, given a countably infinite set $\mathcal{TV}$ of type variables $\alpha, \beta, \gamma, \ldots$, we define by mutual induction:

$$(\textbf{Types})\ \sigma, \tau, \rho \coloneqq \alpha \in \mathcal{TV} \mid \mathcal{M} \mid \mathcal{M} \Rightarrow \sigma$$
$$(\textbf{Multitypes})\ \mathcal{M}, \mathcal{N} \coloneqq [\sigma_i]_{i\in I} \ \text{ where } I \text{ is a finite set}$$

Type environments are denoted by Γ or Δ and we use the usual notations from [5]. The rules of system $\mathcal{B}$ are presented in Fig. 1. Finally, we write $\Pi \triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ when Π is a derivation in system $\mathcal{B}$ with conclusion $\Gamma \vdash t : \sigma$, and $\triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ if there exists some derivation $\Pi \triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$.

As expected, the typing is preserved along both reduction and expansion.

Lemma 1 (Subject Reduction and Expansion for system $\mathcal{B}$ [9, 10]). *Let $t, u \in \Lambda_!$ such that $t \to_{\mathrm{F}} u$, then $\triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$ if and only if $\triangleright_{\mathcal{B}} \Gamma \vdash u : \sigma$.*

Moreover, the type system is sound (all typable terms surface-normalize) and complete (all the surface-normalizing terms are typable).

Lemma 2 (Soundness and Completeness for system $\mathcal{B}$ [9, 10]). *Let $t \in \Lambda_!$, then t has a clash free* S*-normal form if and only if $\triangleright_{\mathcal{B}} \Gamma \vdash t : \sigma$.*

3 The (Distant) Call-by-Value Calculus

Syntax. The *call-by-value λ-calculus* (called dCBV here) is specified using ES and action at a distance [4]. The sets Λ of **terms** and Υ of **values** are inductively defined below:

$$(\textbf{Terms})\quad t, u, s \Coloneqq v \mid \lambda x.t \mid t[x\backslash u] \qquad (\textbf{Values})\quad v \Coloneqq x \in \mathcal{X} \mid \lambda x.t$$

Reductions are driven by the notions of dCBV **list contexts** ($\mathtt{L_V}$)—which allow actions at a *distance*—dCBV **surface contexts** ($\mathtt{S_V}$) and dCBV **full contexts** ($\mathtt{F_V}$):

$$\begin{array}{rrcl} \textbf{(dCBV List Ctxts)} & \mathtt{L_V} & ::= & \diamond \mid \mathtt{L_V}[x\backslash t] \\ \textbf{(dCBV Surface Ctxts)} & \mathtt{S_V} & ::= & \diamond \mid \mathtt{S_V}\, u \mid t\, \mathtt{S_V} \mid \mathtt{S_V}[x\backslash u] \mid t[x\backslash \mathtt{S_V}] \\ \textbf{(dCBV Full Ctxts)} & \mathtt{F_V} & ::= & \diamond \mid \mathtt{F_V}\, u \mid t\, \mathtt{F_V} \mid \lambda x.\mathtt{F_V} \mid \mathtt{F_V}[x\backslash u] \mid t[x\backslash \mathtt{F_V}] \end{array}$$

Operational Semantics. The following **rewrite rules** are the base components of our reductions.

$$\mathtt{L_V}\langle \lambda x.t\rangle\, u \;\mapsto_{\mathtt{dB}}\; \mathtt{L_V}\langle t[x\backslash u]\rangle \qquad\qquad t[x\backslash \mathtt{L_V}\langle v\rangle] \;\mapsto_{\mathtt{sV}}\; \mathtt{L_V}\langle t\{x\backslash v\}\rangle$$

The **surface** (resp. **full**) **reduction** $\rightarrow_{\mathrm{S_V}}$ (resp. $\rightarrow_{\mathrm{F_V}}$) is the defined as the union of the closure of the relations $\{\mathtt{dB}, \mathtt{sV}\}$ by surface (resp. full) contexts. Finally, we denote by $\rightarrow^*_{\mathrm{F_V}}$ (resp. $\rightarrow^*_{\mathrm{S_V}}$) the reflexive transitive closure of $\rightarrow_{\mathrm{F_V}}$ (resp. $\rightarrow_{\mathrm{S_V}}$).

Embedding. A first dCBV-embedding was introduced in [9, 10]. It allows to simulate dCBV with dBang (*i.e.* if $t \rightarrow^*_{\mathrm{S_V}} u$ (resp. $t \rightarrow^*_{\mathrm{F_V}} u$) then $t^{\mathtt{v}} \rightarrow^*_{\mathrm{S}} u^{\mathtt{v}}$ (resp. $t^{\mathtt{v}} \rightarrow^*_{\mathrm{F}} u^{\mathtt{v}}$)). However, the converse known as reverse simulation is false (see Fig.1 in [6]). A novel dCBV embedding was introduced in [6] to resolve this issue. It is defined as follows:

Definition 3. *The dCBV-embedding $\cdot^{\mathtt{v}'} : \Lambda \rightarrow \Lambda_!$ is defined as follows:*

$$\begin{aligned} x^{\mathtt{v}'} &:= !x \\ (\lambda x.t)^{\mathtt{v}'} &:= !\lambda x.!t^{\mathtt{v}'} \\ (tu)^{\mathtt{v}'} &:= \begin{cases} \mathtt{der}\Big(\mathtt{L}\langle s\rangle\, u^{\mathtt{v}'}\Big) & \textit{if } t^{\mathtt{v}'} = \mathtt{L}\langle !s\rangle \\ \mathtt{der}\Big(\mathtt{der}\Big(t^{\mathtt{v}'}\Big)\, u^{\mathtt{v}'}\Big) & \textit{otherwise} \end{cases} \\ (t[x\backslash u])^{\mathtt{v}'} &:= t^{\mathtt{v}'}[x\backslash u^{\mathtt{v}'}] \end{aligned}$$

The main difference can be found in the abstraction case where two ! are used instead of one as in [9, 10, 16]. As intended, surface and full reductions sequences are preserved both ways.

Lemma 4 (dCBV Simulation and Reverse Simulation [6]).
Let $t, u \in \Lambda$, then $t \rightarrow^*_{\mathrm{S_V}} u$ (resp. $t \rightarrow^*_{\mathrm{F_V}} u$) if and only if $t^{\mathtt{v}'} \rightarrow^*_{\mathrm{S}} u^{\mathtt{v}'}$ (resp. $t^{\mathtt{v}'} \rightarrow^*_{\mathrm{F}} u^{\mathtt{v}'}$).

Quantitative Type System. A quantitative type system called system $\mathcal{V}$ have been introduced in [9, 10] to characterize $\mathrm{S_V}$-normalization. As such, typing is preserved throughout reduction sequences and by the embedding $\cdot^{\mathtt{v}}$ [9, 10]. However, this preservation fails with the new embedding $\cdot^{\mathtt{v}'}$. For example, the term $\lambda x.\Omega$ is only typable by $(\emptyset; [\,])$ in system $\mathcal{V}$ while its image $(\lambda x.\Omega)^{\mathtt{v}'} = !\lambda x.!\Omega^{\mathtt{v}'}$ is also typable by $(\emptyset; [[\,] \Rightarrow [\,]])$ in system $\mathcal{B}$.

To address this issue, we now introduce a new quantitative typing system called system $\mathcal{V}'$. The rules of typing system $\mathcal{V}'$ are presented in Fig. 2. The main differences between system $\mathcal{V}$ and $\mathcal{V}'$ is the addition of a new rule called

$$\frac{}{x:\mathcal{M}\vdash x:\mathcal{M}}\ (var)\qquad \frac{\Gamma\vdash t:[\mathcal{M}\Rightarrow[\sigma]]\quad \Delta\vdash u:\mathcal{M}}{\Gamma+\Delta\vdash t\,u:\sigma}\ (app)\qquad \frac{(\Gamma_i\vdash t:\sigma_i)_{i\in I}}{+_{i\in I}\Gamma_i\Vdash t:[\sigma_i]_{i\in I}}\ (frz)$$

$$\frac{\Gamma,x:\mathcal{M}\vdash t:\sigma\quad \Delta\vdash u:\mathcal{M}}{\Gamma+\Delta\vdash t[x\backslash u]:\sigma}\ (es)\qquad \frac{(\Gamma_i,x:\mathcal{M}_i\Vdash t:\sigma_i)_{i\in I}}{+_{i\in I}\Gamma_i\vdash \lambda x.t:[\mathcal{M}_i\Rightarrow\sigma_i]_{i\in I}}\ (abs)$$

Fig. 2: Type System $\mathcal{V}'$ for the `dCBV`-calculus.

(frz) which resembles the rule (bg) from system $\mathcal{B}$. This rule allows to introduce untyped subterms. However, it can only be used in the body of abstractions (thanks to the auxiliary symbol $\Vdash$) and acts as a counterpart to the additional ! in the abstraction case of the novel embedding.

As expected, the typing is preserved by the embedding.

Theorem 5 (Typing Preservation).
Let $t\in\Lambda$, then $\triangleright_{\mathcal{V}'}\Gamma\vdash t:\sigma$ if and only if $\triangleright_{\mathcal{B}}\Gamma\vdash t^{\mathtt{v}'}:\sigma$.

Proof. By induction on t.

Using simulation, we can then deduce, *for free*, that the typing is preserved along full reduction and expansion sequences.

Theorem 6 (Subject Reduction and Expansion of system $\mathcal{V}'$).
Let $t,u\in\Lambda$ such that $t\to_{\mathrm{F}_{\mathtt{v}}}u$, then $\triangleright_{\mathcal{V}'}\Gamma\vdash t:\sigma$ if and only if $\triangleright_{\mathcal{V}'}\Gamma\vdash u:\sigma$.

Proof. Let $t,u\in\Lambda$ such that $t\to_{\mathrm{F}_{\mathtt{v}}}u$. Then $t^{\mathtt{v}'}\to^*_{\mathrm{F}}u^{\mathtt{v}'}$. Using Thm. 6, one deduces that $\triangleright_{\mathcal{B}}\Gamma\vdash t^{\mathtt{v}'}:\sigma$ iff $\triangleright_{\mathcal{B}}\Gamma\vdash u^{\mathtt{v}'}:\sigma$ and therefore $\triangleright_{\mathcal{V}'}\Gamma\vdash t:\sigma$ iff $\triangleright_{\mathcal{V}'}\Gamma\vdash u:\sigma$ using Thm. 5.

Moreover, using surface simulation and reverse simulation, we get *for free* that system $\mathcal{V}'$ is sound and complete.

Theorem 7 (Soundness and Completeness of system $\mathcal{V}'$).
Let $t\in\Lambda$, then t has a $\mathrm{S}_{\mathtt{v}}$-normal form if and only if $\triangleright_{\mathcal{V}'}\Gamma\vdash t:\sigma$.

Proof. Let $t\in\Lambda$, then:
t has a $\mathrm{S}_{\mathtt{v}}$-nf $\overset{Lm.4}{\Leftrightarrow}$ $t^{\mathtt{v}'}$ has a clash free S-nf $\overset{Th.2}{\Leftrightarrow}$ $\triangleright_{\mathcal{B}}\Gamma\vdash t^{\mathtt{v}'}:\sigma$ $\overset{Th.5}{\Leftrightarrow}$ $\triangleright_{\mathcal{V}'}\Gamma\vdash t:\sigma$. □

4 Conclusion and Future Work

In this paper, we introduced an alternative intersection type system for `dCBV` and we have proved that it is preserved by the `dCBV`-embedding into `dBang`. Moreover, we proved that it is sound and complete, and that subject reduction and expansion holds. Still, we would like to further study the preservation of more adanced problems such as *inhabitation* [5] or the *typing equivalence* generated by this typing system [3] and relate them to the usual typing system used for `dCBV`. Finally, we would like to better understand how this changes affect the different notions of *Böhm approximants* and *trees* from the literature [5, 17, 7].

References

1. Accattoli, B., Kesner, D.: The structural *lambda*-calculus. In: Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Proceedings. Lecture Notes in Computer Science, vol. 6247, pp. 381–395. Springer (2010). https://doi.org/10.1007/978-3-642-15205-4_30
2. Accattoli, B., Kesner, D.: Preservation of strong normalisation modulo permutations for the structural lambda-calculus. Logical Methods in Computer Science **8**(1) (2012). https://doi.org/10.2168/LMCS-8(1:28)2012
3. Accattoli, B., Lancelot, A.: Light genericity. In: Kobayashi, N., Worrell, J. (eds.) Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14575, pp. 24–46. Springer (2024). https://doi.org/10.1007/978-3-031-57231-9_2, https://doi.org/10.1007/978-3-031-57231-9_2
4. Accattoli, B., Paolini, L.: Call-by-value solvability, revisited. In: Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Proceedings. Lecture Notes in Computer Science, vol. 7294, pp. 4–16. Springer (2012). https://doi.org/10.1007/978-3-642-29822-6_4
5. Arrial, V., Guerrieri, G., Kesner, D.: Quantitative inhabitation for different lambda calculi in a unifying framework. Proceedings of the ACM on Programming Languages **7**(POPL), 1483–1513 (2023). https://doi.org/10.1145/3571244
6. Arrial, V., Guerrieri, G., Kesner, D.: The benefits of diligence. In: 12th International Joint Conference on Automated Reasoning (IJCAR), 2024, Proceedings (2024)
7. Arrial, V., Guerrieri, G., Kesner, D.: Genericity through stratification. CoRR **abs/2401.12212** (2024). https://doi.org/10.48550/ARXIV.2401.12212, https://doi.org/10.48550/arXiv.2401.12212
8. Barendregt, H.P.: The Lambda Calculus: Its Syntax and Semantics, Studies in logic and the foundations of mathematics, vol. 103. North-Holland (1984)
9. Bucciarelli, A., Kesner, D., Ríos, A., Viso, A.: The Bang Calculus Revisited. In: Functional and Logic Programming, vol. 12073, pp. 13–32. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-59025-3_2
10. Bucciarelli, A., Kesner, D., Ríos, A., Viso, A.: The bang calculus revisited. Information and Computation **293**, 105047 (Aug 2023). https://doi.org/10.1016/j.ic.2023.105047
11. de Carvalho, D.: Sémantiques de la logique linéaire et temps de calcul. Ph.D. thesis, Université Aix-Marseille II (2007)
12. Ehrhard, T., Guerrieri, G.: The Bang Calculus: An untyped lambda-calculus generalizing call-by-name and call-by-value. In: Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, PPDP'16. pp. 174–187. ACM (2016). https://doi.org/10.1145/2967973.2968608
13. Faggian, C., Guerrieri, G.: Factorization in call-by-name and call-by-value calculi via linear logic. In: Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Proceedings. Lecture Notes in Computer Science, vol. 12650, pp. 205–225. Springer (2021). https://doi.org/10.1007/978-3-030-71995-1_11
14. Gardner, P.: Discovering needed reductions using type theory. In: Hagiya, M., Mitchell, J.C. (eds.) Theoretical Aspects of Computer Software. pp. 555–574. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)

15. Girard, J.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987). https://doi.org/10.1016/0304-3975(87)90045-4
16. Guerrieri, G., Manzonetto, G.: The Bang Calculus and the Two Girard's Translations. Electronic Proceedings in Theoretical Computer Science **292**, 15–30 (Apr 2019). https://doi.org/10.4204/EPTCS.292.2
17. Kerinec, A., Manzonetto, G., Pagani, M.: Revisiting call-by-value böhm trees in light of their taylor expansion. Log. Methods Comput. Sci. **16**(3) (2020), https://lmcs.episciences.org/6638
18. Kesner, D., Viso, A.: Encoding tight typing in a unified framework. In: 30th EACSL Annual Conference on Computer Science Logic, CSL 2022. LIPIcs, vol. 216, pp. 27:1–27:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.CSL.2022.27
19. Levy, P.B.: Call-by-push-value: A subsuming paradigm. In: Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1581, pp. 228–242. Springer (1999). https://doi.org/10.1007/3-540-48959-2_17
20. Plotkin, G.D.: Call-by-name, call-by-value and the λ-calculus. Theoretical Computer Science **1**(2), 125–159 (Dec 1975). https://doi.org/10.1016/0304-3975(75)90017-1