Encoding Tight Typing in a Unified Framework

Delia Kesner
Université de Paris, CNRS, IRIF, France
Institut Universitaire de France (IUF), France

Andrés Viso
Inria, Paris, France

Abstract

This paper explores how the intersection type theories of call-by-name (CBN) and call-by-value (CBV) can be *unified* in a more general framework provided by call-by-push-value (CBPV). Indeed, we propose *tight* type systems for CBN and CBV that can be both encoded in a unique *tight* type system for CBPV. All such systems are quantitative, *i.e.* they provide *exact* information about the length of normalization sequences to normal form as well as the size of these normal forms. Moreover, the length of reduction sequences are discriminated according to their multiplicative and exponential nature, a concept inherited from linear logic. Last but not least, it is possible to extract quantitative measures for CBN and CBV from their corresponding encodings in CBPV.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Call-by-Push-Value, Call-by-Name, Call-by-Value, Intersection Types

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.27

Related Version *Full Version*: `https://arxiv.org/abs/2105.00564`

Funding This work has been partially supported by IRP SINFIN.

Acknowledgements We are specially grateful to G. Guerrieri for fruitful discussions. We also thank B. Accattoli, A. Bucciarelli, and A. Ríos for constructive remarks.

1 Introduction

Every programming language implements a particular evaluation strategy which specifies when and how parameters are evaluated during function calls. For example, in **call-by-value (CBV)**, the argument is evaluated before being passed to the function, while in **call-by-name (CBN)** the argument is substituted directly into the function body, so that the argument may never be evaluated, or may be re-evaluated several times. CBN and CBV have always been studied independently, by developing different techniques for one and the other, until the remarkable observation that they are two different instances of a more general framework introduced by Girard's **Linear Logic (LL)**. Their (logical) duality ("CBN is *dual* to CBV") was understood later [18, 17]. And their rewriting semantics were finally *unified* by the **call-by-push-value (CBPV)** paradigm – introduced by P.B. Levy [41, 42] – a formalism being able to capture different functional languages/evaluation strategies.

A typical aspect that one wants to compare between two different evaluation strategies is the *number of steps* that are necessary to get a result. Such numbers are extracted from different models of computation and should be then measured by *compatible* instruments, either by means of common quantitative tools, or by a precise transformation between them[1]. Thus, we provide a *uniform* tool to measure quantitative information extracted from the evaluation of programs in different programming languages (namely CBN and CBV).

[1] Think for example about cm and inches.

More concretely, we introduce typing systems capturing **qualitative** and **quantitative** information of programs. From a qualitative point of view, the typing systems characterize termination of programs, *i.e.* a program p is typable if and only if p terminates. From a quantitative point of view, the typing systems provide **exact/tight** information about the **number of steps** needed to get a result, as well as the **size** of this result. All these questions are addressed in the general framework of CBPV, being able to encode CBN and CBV, both from a static and a dynamic point of view. Dynamically, CBN and CBV evaluation strategies are known to be encodable by the rewriting semantics of CBPV. Statically, we define tight typing systems providing quantitative information for CBN and CBV, that can be seen as particular cases of the tight quantitative typing system behind the unified framework of CBPV. We now explain all these concepts in more detail.

Quantitative Types. Quantitative typing systems are often specified by non-idempotent intersection types inspired by the relational semantics of LL [30, 11], as pioneered by [28]. This connection makes non-idempotent types not only a qualitative typing tool to reason about programming languages, but mainly a quantitative one, being able to specify properties related to the consumption of resources, a remarkable investigation pioneered by the seminal de Carvalho's PhD thesis [19] (see also [21]). Thus, qualitatively, a non-idempotent typing system is able to fully **characterise** normalisation, in the sense that a term t is typable if and only if t is normalising. More interestingly, quantitative typing systems also provide **upper bounds**, in the sense that the length of any reduction sequence from t to normal form *plus* the size of this normal form is *bounded* by the size of the type derivation of t. Therefore, typability characterises normalisation in a qualitative as well as in a quantitative way, but only provides upper bounds. Several papers explore bounded measures of non-idempotent types for different higher order languages. Some references are [24, 31, 4, 3, 35, 14, 20, 33, 23, 22, 13, 44].

In this satisfactory enough? A major observation concerning β-reduction in λ-calculus is that the size of normal forms can be *exponentially* bigger than the number of steps needed to reach these normal forms. This means that bounding the sum of these two integers *at the same* time is too rough, and not very relevant from a quantitative point of view. Fortunately, it is possible to extract better (*i.e.* independent and exact) measures from a non-idempotent intersection type system. A crucial point to obtain **exact measures**, instead of upper bounds, is to consider *minimal* type derivations [19, 9, 23]. Therefore, *upper bounds* for time *plus* size can be refined into *independent exact measures* for time *and* size [1]. More precisely, the quantitative typing systems are now equipped with constants and *counters*, together with an appropriate notion of **tightness**, which encodes minimality of type derivations. For any *tight* type derivation Φ of a term t ending with (independent) counters (b, s), it is now possible to show that t is normalisable in b steps and its normal form has size s, so that the type system is able to *guess* the number of steps to normal form as well as the size of this normal form. The opposite direction also holds: if t normalises in b steps to a normal form size s, then it is possible to tightly type t by using (independent) counters (b, s).

In this paper we design tight quantitative type systems that are also capable to discriminate between **multiplicative** and **exponential** evaluation steps to normal form, two conceptual notions coming from LL: *multiplicative* steps are essentially those that (linearly) reconfigure proofs/terms/programs, while *exponential* steps are the only ones that are potentially able to erase/duplicate other objects. As a consequence, for any *tight* type derivation Φ of t ending with (independent) counters (m, e, s), the term t is normalisable in m multiplicative and e exponential steps to a normal form having size s. The opposite direction also holds.

Call-by-push-value. CBPV extends the λ-calculus with two primitives `thunk` and `force` distinguishing between *values* and *computations*: the former freezes the execution of a term (*i.e.* turns a computation into a value) while the latter fires again a frozen term (*i.e.* turns a value into a computation). These primitives allow to capture the duality between CBN and CBV by conveniently labelling a λ-term with `thunk`/`force` to pause/resume the evaluation of a subterm. Thus, CBPV provides a *unique* and *general* formalism capturing different functional strategies, and allowing to *uniformly* study operational and denotational semantics of different programming languages through a single tool.

In this paper we model the CBPV paradigm by using the $\lambda!$-calculus [12], based on the *bang calculus* introduced in [26], which in turn extends ideas by T. Ehrhard [25]. The granularity of the $\lambda!$-calculus, expressed with both **explicit substitutions** and **reduction at a distance** (details in Sec. 5), clearly allows to differentiate between *multiplicative* and *exponential* steps, as in LL. The corresponding CBN and CBV strategies also follow this pattern: it is possible to distinguish the multiplicative steps that only reconfigure pieces of syntax, from the exponential steps used to implement erasure and duplication of terms.

Contributions. We first define deterministic strategies for CBN and CBV that are able to discriminate between multiplicative and exponential steps (Sec. 2).

We then formulate tight typing systems for both CBN (Sec. 3) and CBV (Sec. 4), called respectively $\mathcal{N}$ and $\mathcal{V}$. System $\mathcal{N}$ is a direct extension of Gardner's system [28], while system $\mathcal{V}$ is completely new, and constitutes one of the major contributions of this paper. A key feature of system $\mathcal{V}$ is its ability to distinguish between the two different roles that variables may play in CBV depending on the context where they are placed, *i.e.* to be a placeholder for a *value* or the head of a *neutral* term. We show that both systems implement independent measures for time and size, and that they are quantitatively sound and complete. More precisely, we show that tight (*i.e.* minimal) typing derivations in such systems (exactly) quantitatively characterise normalisation, *i.e.* if Φ is a *tight* type derivation of t in system $\mathcal{N}$ (resp. $\mathcal{V}$), ending with counters (m, e, s), then there exists a CBN (resp. CBV) normal form p of size s such that t reduces to p by using exactly m multiplicative steps and e exponential steps. The converse, giving quantitative completeness of the approach, also holds.

Sec. 5 recalls the bang calculus at a distance $\lambda!$ and Sec. 6 presents its associated tight type system $\mathcal{B}$, together with their respective quantitative sound and complete properties. The untyped CBN/CBV translations into $\lambda!$ are recalled in Sec. 7, while the typed translations are defined and discussed in Sec. 8, the other major contribution of this work. Through these typed encodings, the counters of the source and target derivations are related. This makes it possible to give the precise cost of our typed translations, as well as to extract quantitative measures for CBN and CBV from their corresponding encodings in CBPV.

Detailed proofs can be found in [37].

Related Work. For CBN, non-idempotent types were introduced by [28], their quantitative power was extensively studied in [19, 20], and their tight extensions in [9, 1]. For CBV, non-idempotent types were introduced in [24], and extensively studied, *e.g.* [31, 3]. A tight extension being able to count reduction steps was recently defined in [4, 40] for a special version of (closed) CBV, but it is not clear how this could be encoded in a linear logic based CBPV framework. Another non-idempotent type system was also recently introduced for CBV [43, 34], it is not tight and does not translate to CBPV.

A (non-tight) quantitative type system for the bang calculus, based on a relational model, can be found in [32]. Another relational model that can be seen as a non-tight system for CBPV was introduced by [16]. Following ideas in [19, 9, 1], a type system $\mathcal{E}$ was

proposed in [12] to fully exploit tight quantitative aspects of the $\lambda!$-calculus: *independent exact measures* for time and size are guessed by the types system. However, no relation between tightness for CBN/CBV and tightness for the $\lambda!$-calculus are studied in *op.cite.* This papers fills this gap.

The discrimination between multiplicative and exponential steps by means of tight quantitative types can be found *e.g.* in CBN [1], call-by-need [4], and languages with pattern-matching primitives [7].

2 Call-by-Name and Call-by-Value

This section introduces the CBN and CBV specifications being able to distinguish between multiplicative and exponential steps, as in linear logic. Given a countably infinite set $\mathcal{X}$ of variables $x, y, z, \ldots$, we consider the following grammars for terms ($\mathcal{T}_\lambda$), values and contexts:

$$\begin{array}{rrcl}
\textbf{\textit{(Terms)}} & t,u,r & ::= & v \mid t\,u \mid t[x\backslash u] \\
\textbf{\textit{(Values)}} & v & ::= & x \in \mathcal{X} \mid \lambda x.t \\
\textbf{\textit{(Contexts)}} & \mathtt{C} & ::= & \mathtt{L} \mid \mathtt{N} \mid \mathtt{V} \\
\textbf{\textit{(List Contexts)}} & \mathtt{L} & ::= & \Box \mid \mathtt{L}[x\backslash t] \\
\textbf{\textit{(CBN Contexts)}} & \mathtt{N} & ::= & \Box \mid \mathtt{N}\,t \mid \lambda x.\mathtt{N} \mid \mathtt{N}[x\backslash u] \\
\textbf{\textit{(CBV Contexts)}} & \mathtt{V} & ::= & \Box \mid \mathtt{V}\,t \mid t\,\mathtt{V} \mid \mathtt{V}[x\backslash u] \mid t[x\backslash \mathtt{V}]
\end{array}$$

A terms of the form $t[x\backslash u]$ is a ***closure***, and $[x\backslash u]$ an ***explicit substitution*** (ES). Special terms are $\mathtt{I} = \lambda z.z$, $\mathtt{K} = \lambda x.\lambda y.x$, $\Delta = \lambda x.x\,x$, and $\Omega = \Delta\,\Delta$. We use $\mathtt{C}\langle t\rangle$ for the term obtained by replacing the hole $\Box$ of $\mathtt{C}$ by t. ***Free*** and ***bound*** variables, as well as α-conversion, are defined as expected. In particular, $\mathtt{fv}(t[x\backslash u]) \stackrel{def}{=} \mathtt{fv}(t)\setminus\{x\}\cup\mathtt{fv}(u)$, $\mathtt{fv}(\lambda x.t) \stackrel{def}{=} \mathtt{fv}(t)\setminus\{x\}$, $\mathtt{bv}(t[x\backslash u]) \stackrel{def}{=} \mathtt{bv}(t)\cup\{x\}\cup\mathtt{bv}(u)$ and $\mathtt{bv}(\lambda x.t) \stackrel{def}{=} \mathtt{bv}(t)\cup\{x\}$. The notation $t\,\{x\backslash u\}$ is used for the (capture-free) ***meta-level*** substitution operation, defined, as usual, modulo α-conversion. Special predicates are used to distinguish different kinds of terms surrounded by ES: $\mathsf{abs}(t)$ iff $t = \mathtt{L}\langle\lambda x.u\rangle$, $\mathsf{app}(t)$ iff $t = \mathtt{L}\langle r\,u\rangle$ and $\mathsf{var}(t)$ iff $t = \mathtt{L}\langle x\rangle$. Finally, $\mathsf{val}(t)$ iff $\mathsf{abs}(t)$ or $\mathsf{var}(t)$.

As mentioned in the introduction, our aim is to count the reduction steps by distinguishing their multiplicative and exponential nature. To achieve this, the standard specifications of CBN/CBV are not adequate, so we need to consider alternative appropriate definitions [6] making use of the following three different rewriting rules:

$$\begin{array}{rrcl}
(\textbf{d}\text{istant }\textbf{B}\text{eta}) & \mathtt{L}\langle\lambda x.t\rangle\,u & \mapsto_{\mathtt{dB}} & \mathtt{L}\langle t[x\backslash u]\rangle \\
(\textbf{s}\text{ubstitute term}) & t[x\backslash u] & \mapsto_{\mathtt{sn}} & t\,\{x\backslash u\} \\
(\textbf{s}\text{ubstitute value}) & t[x\backslash \mathtt{L}\langle v\rangle] & \mapsto_{\mathtt{sv}} & \mathtt{L}\langle t\,\{x\backslash v\}\rangle
\end{array}$$

Rule $\mathtt{dB}$ fires β-reduction *at a distance* by combining the two more elementary rules: $(\lambda x.t)\,u \mapsto t[x\backslash u]$ and $\mathtt{L}\langle t\rangle\,u \mapsto \mathtt{L}\langle t\,u\rangle$, where the second one is a structural/permutation rule pushing out ES that may block β-redexes. Rule $\mathtt{sn}$ implements standard substitution, while $\mathtt{sv}$ restricts substitution to values and acts *at a distance* by combining the two more elementary rules: $t[x\backslash v] \mapsto t\,\{x\backslash v\}$ and $t[x\backslash \mathtt{L}\langle v\rangle] \mapsto \mathtt{L}\langle t[x\backslash v]\rangle$. The ***call-by-name*** reduction relation $\to_{\mathtt{n}}$ is the closure by contexts $\mathtt{N}$ of the rules $\mathtt{dB}$ and $\mathtt{sn}$, while the ***call-by-value*** reduction relation $\to_{\mathtt{v}}$ is the closure by contexts $\mathtt{V}$ of the rules $\mathtt{dB}$ and $\mathtt{sv}$. Equivalently,

$$\to_{\mathtt{n}} := \mathtt{N}(\mapsto_{\mathtt{dB}} \cup \mapsto_{\mathtt{sn}}) \quad \text{and} \quad \to_{\mathtt{v}} := \mathtt{V}(\mapsto_{\mathtt{dB}} \cup \mapsto_{\mathtt{sv}})$$

The resulting CBN/CBV formulations are now based on distinguished ***multiplicative*** (*cf.* $\mathtt{dB}$) and ***exponential*** (*cf.* $\mathtt{sn}$ and $\mathtt{sv}$) steps, called resp. $\mathtt{m}$-steps and $\mathtt{e}$-steps, thus inheriting the nature of cut elimination rules in LL. Notice that the number of $\mathtt{m}$ and $\mathtt{e}$-steps in a

normalization sequence is not always the same: *e.g.* $x[x\backslash y] \to_{\mathtt{n}} y$ has only one $\mathtt{e}$-step, and $(\lambda x.x)\,(z\,\mathtt{I}) \to_{\mathtt{v}} x[x\backslash z\,\mathtt{I}]$ has only one $\mathtt{m}$-step. We write $t \not\to_{\mathtt{n}}$ (resp. $t \not\to_{\mathtt{v}}$), and call t an ***$\mathtt{n}$-normal form*** (resp. ***$\mathtt{v}$-normal form***), if t cannot be reduced by means of $\to_{\mathtt{n}}$ (resp. $\to_{\mathtt{v}}$). Both CBN and CBV are non-deterministic: $t \to_{\mathtt{n}} u$ and $t \to_{\mathtt{n}} s$ does not necessarily implies $u = s$. But both calculi enjoy confluence, notably because their rules are orthogonal [39, 38]. Moreover, in each calculus, it is easy to show that any two different reduction paths to normal form have the same number of multiplicative and exponential steps.

CBN is to be understood as *head* reduction [8], *i.e.* reduction does not take place in arguments of applications, while CBV corresponds to *open* CBV reduction [6, 2], *i.e.* reduction does not take place inside abstractions. The sets of $\mathtt{n}$/$\mathtt{v}$-normal forms can be alternatively characterised by the following grammars [12]:

$$\begin{array}{ll} \textbf{\textit{(CBN Neutral)}} & \mathsf{ne}_{\mathtt{n}} ::= x \in \mathcal{X} \mid \mathsf{ne}_{\mathtt{n}}\, t \\ \textbf{\textit{(CBN Normal)}} & \mathsf{no}_{\mathtt{n}} ::= \lambda x.\mathsf{no}_{\mathtt{n}} \mid \mathsf{ne}_{\mathtt{n}} \end{array} \qquad \begin{array}{ll} \textbf{\textit{(CBV Variable)}} & \mathsf{vr}_{\mathtt{v}} ::= x \in \mathcal{X} \mid \mathsf{vr}_{\mathtt{v}}[x\backslash \mathsf{ne}_{\mathtt{v}}] \\ \textbf{\textit{(CBV Neutral)}} & \mathsf{ne}_{\mathtt{v}} ::= \mathsf{vr}_{\mathtt{v}}\,\mathsf{no}_{\mathtt{v}} \mid \mathsf{ne}_{\mathtt{v}}\,\mathsf{no}_{\mathtt{v}} \mid \mathsf{ne}_{\mathtt{v}}[x\backslash \mathsf{ne}_{\mathtt{v}}] \\ \textbf{\textit{(CBV Normal)}} & \mathsf{no}_{\mathtt{v}} ::= \lambda x.t \mid \mathsf{vr}_{\mathtt{v}} \mid \mathsf{ne}_{\mathtt{v}} \mid \mathsf{no}_{\mathtt{v}}[x\backslash \mathsf{ne}_{\mathtt{v}}] \end{array}$$

In contrast to CBN, variables are left out of the definition of neutral terms for the CBV case, since they are now considered as values. However, even if CBV variables are not neutral terms, neutral terms are necessarily headed by a variable, so that variables play a double role which is difficult to be distinguished by means of an intersection type system. We will come back to this point in Sec. 4. Excluding variables from the set of values brings a remarkable speed up in implementations of CBV [40], but goes beyond the logical Girard's translation of CBV into LL, which is the main topic of this paper. Our chosen approach allows both CBN and CBV neutral terms to translate to neutral terms of the λ!-calculus (*cf.* Sec. 5).

Deterministic Strategies for CBN and CBV. As a technical tool, in order to count the reduction steps of CBN/CBV we first fix a deterministic version for them. The reduction relation $\to_{\mathtt{dn}}$ is a deterministic version of $\to_{\mathtt{n}}$ defined as:

$$\frac{}{\mathtt{L}\langle\lambda x.t\rangle\, u \to_{\mathtt{dn}} \mathtt{L}\langle t[x\backslash u]\rangle} \qquad \frac{}{t[x\backslash u] \to_{\mathtt{dn}} t\,\{x\backslash u\}} \qquad \frac{t \to_{\mathtt{dn}} s \text{ and } \neg\mathsf{abs}(t)}{t\,u \to_{\mathtt{dn}} s\,u} \qquad \frac{t \to_{\mathtt{dn}} s}{\lambda x.t \to_{\mathtt{dn}} \lambda x.u}$$

Similarly, the reduction relation $\to_{\mathtt{dv}}$ is a deterministic version of $\to_{\mathtt{v}}$ defined as:

$$\frac{}{\mathtt{L}\langle\lambda x.t\rangle\, u \to_{\mathtt{dv}} \mathtt{L}\langle t[x\backslash u]\rangle} \qquad \frac{}{t[x\backslash \mathtt{L}\langle v\rangle] \to_{\mathtt{dv}} \mathtt{L}\langle t\,\{x\backslash v\}\rangle} \qquad \frac{t \to_{\mathtt{dv}} s \quad \neg\mathsf{abs}(t)}{t\,u \to_{\mathtt{dv}} s\,u}$$

$$\frac{t \to_{\mathtt{dv}} s \quad u \in \mathsf{ne}_{\mathtt{v}} \cup \mathsf{vr}_{\mathtt{v}}}{u\,t \to_{\mathtt{dv}} u\,s} \qquad \frac{t \to_{\mathtt{dv}} s \quad \neg\mathsf{val}(t)}{u[x\backslash t] \to_{\mathtt{dv}} u[x\backslash s]} \qquad \frac{t \to_{\mathtt{dv}} s \quad u \in \mathsf{ne}_{\mathtt{v}}}{t[x\backslash u] \to_{\mathtt{dv}} s[x\backslash u]}$$

As a matter of notation, for $\mathcal{X} \in \{\mathtt{dn}, \mathtt{dv}\}$, we write $t \twoheadrightarrow_{\mathcal{X}}^{(m,e)} u$ if $t \twoheadrightarrow_{\mathcal{X}} u$ using m multiplicative steps and e exponential steps.

The normal forms of the non-deterministic and the deterministic versions of CBN/CBV are the same, in turn characterised by the grammars $\mathsf{no}_{\mathtt{n}}/\mathsf{no}_{\mathtt{v}}$ [12].

▶ **Proposition 1.** *Let $t \in \mathcal{T}_\lambda$. Then, $t \not\to_{\mathtt{n}}$ iff $t \not\to_{\mathtt{dn}}$ iff $t \in \mathsf{no}_{\mathtt{n}}$ and $t \not\to_{\mathtt{v}}$ iff $t \not\to_{\mathtt{dv}}$ iff $t \in \mathsf{no}_{\mathtt{v}}$.*

(Head) CBN ignores reduction inside arguments of applications, while (Open) CBV ignores reduction inside abstractions, then CBN (resp. CBV) normal forms are measured by the following ***$\mathtt{n}$-size*** (resp. ***$\mathtt{v}$-size***) function:

$$\begin{array}{llll} |x|_{\mathtt{n}} := 0 & |\lambda x.t|_{\mathtt{n}} := |t|_{\mathtt{n}} + 1 & |t\,u|_{\mathtt{n}} := |t|_{\mathtt{n}} + 1 & |t[x\backslash u]|_{\mathtt{n}} := |t|_{\mathtt{n}} \\ |x|_{\mathtt{v}} := 0 & |\lambda x.t|_{\mathtt{v}} := 0 & |t\,u|_{\mathtt{v}} := |t|_{\mathtt{v}} + |u|_{\mathtt{v}} + 1 & |t[x\backslash u]|_{\mathtt{v}} := |t|_{\mathtt{v}} + |u|_{\mathtt{v}} \end{array}$$

3 Tight Call-by-Name

We now introduce a tight type system $\mathcal{N}$ for CBN which captures independent exact measures for $\to_{\mathtt{n}}$-reduction sequences. This result is not surprising, since it extends the one in [1] from the pure λ-calculus to our CBN calculus. However, we revisit the op.cit. approach by appropriately splitting reduction into multiplicative and exponential steps, a reformulation necessary to establish a precise correspondence with the tight type system for the $\lambda!$-calculus. In particular, our *counting* mechanism slightly differs from [1] (details below).

In system $\mathcal{N}$ there are two base types: $\mathtt{a}$ types terms whose normal form is an abstraction, and $\mathtt{n}$ types terms whose normal form is CBN neutral. The grammar of types is given by:

$$\begin{array}{rrcl} \textbf{\textit{(Tight Types)}} & \mathtt{tt} & ::= & \mathtt{n} \mid \mathtt{a} \\ \textbf{\textit{(Types)}} & \sigma, \tau & ::= & \mathtt{tt} \mid \mathcal{M} \mid \mathcal{M} \to \sigma \\ \textbf{\textit{(Multitypes)}} & \mathcal{M}, \mathcal{N} & ::= & [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set} \end{array}$$

Multitypes are multisets of types. The ***empty multitype*** is denoted by $[\,]$, $\sqcup$ denotes multitype union, and $\sqsubseteq$ multitype inclusion. Also, $|\mathcal{M}|$ denotes the size of the multitype, thus if $\mathcal{M} = [\sigma_i]_{i \in I}$ then $|\mathcal{M}| = \#(I)$. Notice that the grammar for types slightly differs from [1], in particular types are now allowed to be just multitypes. The main reason to adopt this change is that this (unique) grammar is used for our three formalisms CBN, CBV, and CBPV, changing only the definition of tight types for each case[2].

Typing contexts (or just ***contexts***), written Γ, Δ, are functions from variables to multitypes, assigning the empty multitype to all but a finite set of variables. The domain of Γ is given by $\mathtt{dom}(\Gamma) \stackrel{def}{=} \{x \mid \Gamma(x) \neq [\,]\}$. The ***union of contexts***, written $\Gamma + \Delta$, is defined by $(\Gamma + \Delta)(x) \stackrel{def}{=} \Gamma(x) \sqcup \Delta(x)$. An example is $(x : [\sigma], y : [\tau]) + (x : [\sigma], z : [\tau]) = (x : [\sigma, \sigma], y : [\tau], z : [\tau])$. This notion is extended to several contexts as expected, so that $+_{i \in I}\, \Gamma_i$ denotes a finite union of contexts (particularly the empty context when $I = \emptyset$). We write $\Gamma \setminus\!\setminus x$ for the context $(\Gamma \setminus\!\setminus x)(x) = [\,]$ and $(\Gamma \setminus\!\setminus x)(y) = \Gamma(y)$ if $y \neq x$.

Type judgements have the form $\Gamma \vdash^{(m,e,s)} t : \sigma$, where Γ is a typing context, t is a term, σ is a type, and the ***counters*** (m, e, s) are expected to provide the following information: m (resp. e) indicates the number of multiplicative $\mathtt{m}$-steps (resp. exponential $\mathtt{e}$-steps) to normal form, while s indicates the $\mathtt{n}$-size of this normal form. It is worth noticing that the λ-calculus hides both multiplicative and exponential steps in one single β-reduction rule [1], so that only two counters suffice, one for the number of β-reduction steps, and another for the size of normal forms. Here we want to discriminate between multiplicative/exponential steps, in the sense that the execution of an ES generates an exponential step, but not a multiplicative one. This becomes possible due to the CBN/CBV alternative specifications with ES that we have adopted, and that is why we need three independent counters, in contrast to the λ-calculus.

The type system $\mathcal{N}$ for CBN is given in Fig. 1 (*persistent* rules) and 2 (*consuming* rules). A constructor is consuming (resp. persistent) if it is consumed (resp. not consumed) during $\mathtt{n}$-reduction. For instance, in $\mathtt{K}\,\mathtt{I}\,\Omega$ the two abstractions of $\mathtt{K}$ are consuming, while the abstraction of $\mathtt{I}$ is persistent, and all the other constructors are also consuming, except those of Ω that turns out to be an untyped subterm. The persistent rules (Fig. 1) are those typing persistent constructors, so that none of them increases the first two counters, but only possibly the third one, which contributes to the size of the normal form. The consuming rules (Fig. 2), in contrast, type consuming constructors, so that they may increase the first

[2] In the intersection type literature CBN and CBV do always adopt different grammars.

two counters, contributing to the length of the normalisation sequence. Notice in particular that there are only two rules contributing to the multiplicative/exponential counting:

(1) rule $(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$ types a *consuming* application, meaning that its left-hand side subterm reduces to an abstraction, then causing a multiplicative step (rule $\mathtt{dB}$) [3] followed later by an exponential step; while

(2) rule $(\mathtt{es}_{\mathtt{c}}^{\mathcal{N}})$ types a *consuming* substitution causing an exponential step. In both cases, it is the constructor application/substitution which is considered to be consumed, without any further hypothesis on the form of the subterm that appears inside this consuming constructor (recall that any term can be substituted in CBN). This phenomenon facilitates in particular the identification of exponential steps in CBN, in contrast to CBV and CBPV.

$$\frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{n}}{\Gamma \vdash^{(m,e,s+1)} t\, u : \mathtt{n}}\,(\mathtt{app}_{\mathtt{p}}^{\mathcal{N}}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{tt} \quad \mathsf{tight}(\Gamma(x))}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s+1)} \lambda x.t : \mathtt{a}}\,(\mathtt{abs}_{\mathtt{p}})$$

Figure 1 System $\mathcal{N}$ for the Call-by-Name Calculus: Persistent Typing Rules.

$$\frac{}{x : [\sigma] \vdash^{(0,0,0)} x : \sigma}\,(\mathtt{var}_{\mathtt{c}}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \tau}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s)} \lambda x.t : \Gamma(x) \to \tau}\,(\mathtt{abs}_{\mathtt{c}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : [\sigma_i]_{i\in I} \to \tau \quad (\Delta_i \vdash^{(m_i,e_i,s_i)} u : \sigma_i)_{i\in I}}{\Gamma + \Delta \vdash^{(1+m+_{i\in I} m_i,\, 1+e+_{i\in I} e_i,\, s+_{i\in I} s_i)} t\, u : \tau}\,(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$$

$$\frac{\Gamma; x : [\sigma_i]_{i\in I} \vdash^{(m,e,s)} t : \tau \quad (\Delta_i \vdash^{(m_i,e_i,s_i)} u : \sigma_i)_{i\in I}}{(\Gamma \setminus\!\setminus x) +_{i\in I} \Delta_i \vdash^{(m+_{i\in I} m_i,\, 1+e+_{i\in I} e_i,\, s+_{i\in I} s_i)} t[x\backslash u] : \tau}\,(\mathtt{es}_{\mathtt{c}}^{\mathcal{N}})$$

Figure 2 System $\mathcal{N}$ for the Call-by-Name Calculus: Consuming Typing Rules.

This dichotomy between consuming/persistent constructors has been first used in [36] for the λ and $\lambda\mu$-calculi, and adapted here for our distant versions of CBN/CBV as well as for the λ!-calculus. We write $\rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ if there is a ***(tree) type derivation*** of the judgement $\Gamma \vdash^{(m,e,s)} t : \sigma$ in system $\mathcal{N}$. The term t is typable in system $\mathcal{N}$, or $\mathcal{N}$-typable, iff there is a context Γ, a type σ and counters (m, e, s) such that $\rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$. We use the capital Greek letters $\Phi, \Psi, \ldots$ to name type derivations, by writing for example $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$. As (local) counters of judgements in a given derivation Φ contribute to the global counters of the derivation itself, there is an alternative way to define counters associated to a derivation Φ: the first counter m_Φ is given by the number of rules $(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$ in Φ, the second counter e_Φ is the number of rules $(\mathtt{es}_{\mathtt{c}}^{\mathcal{N}})$ in Φ and finally the third counter s_Φ is the number of rules $(\mathtt{app}_{\mathtt{p}}^{\mathcal{N}})$ and $(\mathtt{abs}_{\mathtt{p}})$ in Φ. We prefer however to systematically write counters in judgements to easy the understanding of the examples and proofs.

A ***multitype*** $[\sigma_i]_{i\in I}$ is ***tight***, written $\mathsf{tight}([\sigma_i]_{i\in I})$, if $\sigma_i \in \mathtt{tt}$ for all $i \in I$. A ***context*** Γ is said to be ***tight*** if it assigns tight multitypes to all variables. A ***type derivation*** $\Phi \rhd_{\mathcal{B}} \Gamma \vdash^{(m,e,s)} t : \sigma$ is ***tight*** if Γ is tight and $\sigma \in \mathtt{tt}$.

The proofs of soundness and completeness related to our CBN type system are respectively based on subject reduction and expansion properties, and they are very similar to those in [1]. The most important point to be mentioned is that system $\mathcal{N}$ is now counting *separately* the multiplicative and exponential steps of $\to_{\mathtt{n}}$-reductions to normal-form.

[3] In both [1] and [36], it is the consuming abstraction which contributes to the multiplicative steps.

▶ **Theorem 2** (Soundness and Completeness).

- *If* $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight, then there exists* p *such that* $p \in \mathsf{no}_{\mathtt{n}}$ *and* $t \twoheadrightarrow_{\mathtt{n}}^{(m,e)} p$ *with* m $\mathtt{m}$*-steps,* e $\mathtt{e}$*-steps, and* $|p|_{\mathtt{n}} = s$.
- *If* $t \twoheadrightarrow_{\mathtt{n}}^{(m,e)} p$ *with* $p \in \mathsf{no}_{\mathtt{n}}$*, then there exists a tight type derivation* $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,|p|_{\mathtt{n}})} t : \sigma$.

Notice that the previous theorem is stated by using the general notion of reduction $\to_{\mathtt{n}}$, but the proofs are done using the deterministic reduction $\to_{\mathtt{dn}}$. The equivalence holds because any two different reduction paths to normal form have the same number of multiplicative and exponential steps, as already remarked.

▶ **Example 3.** Consider $t_0 = \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})$ that $\to_{\mathtt{n}}$-reduces in 2 $\mathtt{m}$-steps and 2 $\mathtt{e}$-steps to the term $z\,\mathtt{I} \in \mathsf{no}_{\mathtt{n}}$, whose $\mathtt{n}$-size is 1:

$$t_0 = \underline{\mathtt{K}\,(z\,\mathtt{I})}\,(\mathtt{I}\,\mathtt{I}) \to_{\mathtt{dB}} \underline{(\lambda y.x)[x\backslash z\,\mathtt{I}]\,(\mathtt{I}\,\mathtt{I})} \to_{\mathtt{dB}} \underline{x[y\backslash \mathtt{I}\,\mathtt{I}][x\backslash z\,\mathtt{I}]} \to_{\mathtt{sn}} \underline{(z\,\mathtt{I})[y\backslash \mathtt{I}\,\mathtt{I}]} \to_{\mathtt{sn}} z\,\mathtt{I}$$

System $\mathcal{N}$ admits a tight type derivation for t_0 with the expected final counter $(2,2,1)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{n}] \vdash^{(0,0,0)} x : \mathtt{n}}\,(\mathtt{var_c})}{x : [\mathtt{n}] \vdash^{(0,0,0)} \lambda y.x : [\,] \to \mathtt{n}}\,(\mathtt{abs_c})}{\vdash^{(0,0,0)} \mathtt{K} : [\mathtt{n}] \to [\,] \to \mathtt{n}}\,(\mathtt{abs_c}) \quad \dfrac{\dfrac{}{z : [\mathtt{n}] \vdash^{(0,0,0)} z : \mathtt{n}}\,(\mathtt{var_c})}{z : [\mathtt{n}] \vdash^{(0,0,1)} z\,\mathtt{I} : \mathtt{n}}\,(\mathtt{app}_{\mathtt{p}}^{\mathcal{N}})}{z : [\mathtt{n}] \vdash^{(1,1,1)} \mathtt{K}\,(z\,\mathtt{I}) : [\,] \to \mathtt{n}}\,(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})}{z : [\mathtt{n}] \vdash^{(2,2,1)} \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I}) : \mathtt{n}}\,(\mathtt{app}_{\mathtt{c}}^{\mathcal{N}})$$

4 Tight Call-by-Value

In this section we introduce a tight system $\mathcal{V}$ for CBV which captures exact measures for $\to_{\mathtt{v}}$-reduction sequences. Here we do not only distinguish length of reduction sequences from size of normal forms, but also discriminate multiplicative from exponential steps. Moreover, we establish a precise relation between the counters in CBV and their counterparts in the $\lambda!$-calculus (*cf.* Sec. 8). This constitutes one of the main contributions of the present work.

The grammar of types of system $\mathcal{V}$ is given by:

(Tight Types)	$\mathtt{tt}$	$::=$	$\mathtt{n} \mid \mathtt{vl} \mid \mathtt{vr}$
(Types)	σ, τ	$::=$	$\mathtt{tt} \mid \mathcal{M} \mid \mathcal{M} \to \sigma$
(Multitypes)	$\mathcal{M}, \mathcal{N}$	$::=$	$[\sigma_i]_{i \in I}$ where I is a finite set

Indeed, apart from the constant $\mathtt{n}$ also used in CBN, we now introduce constants $\mathtt{vl}$ and $\mathtt{vr}$, typing respectively, terms reducing to values; and terms reducing to persistent variables that are not acting as values (*i.e.* they are applied to some argument to produce neutral terms). Notice that we remove the base type $\mathtt{a}$ from CBN, since abstractions are in particular values, so they will be typed with $\mathtt{vl}$. The notions of ***tightness*** for multitypes, contexts and derivations are exactly the same we used for CBN.

A type system for CBV must type variables without knowing yet the future role they are going to play (head of neutral term or value). This is one of the main difficulties behind the definition of such a system. Besides that, the system must also be able to count multiplicative and exponential steps independently. The resulting type system $\mathcal{V}$ for CBV is given by the typing rules in Fig. 3 and 4, distinguishing between persistent and consuming rules resp. As a matter of notation, given a tight type $\mathtt{tt}_0$ we write $\overline{\mathtt{tt}_0}$ to denote a tight type different from $\mathtt{tt}_0$. Thus for instance, $\overline{\mathtt{vr}} \in \{\mathtt{vl}, \mathtt{n}\}$.

$$\frac{}{x : [\mathtt{vr}] \vdash^{(0,0,0)} x : \mathtt{vr}}\,(\mathtt{var_p}) \qquad \frac{}{\vdash^{(0,0,0)} x : \mathtt{vl}}\,(\mathtt{val_p}) \qquad \frac{}{\vdash^{(0,0,0)} \lambda x.t : \mathtt{vl}}\,(\mathtt{abs}_\mathtt{p}^{\mathcal{V}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \overline{\mathtt{vl}} \quad \Delta \vdash^{(m',e',s')} u : \overline{\mathtt{vr}}}{\Gamma + \Delta \vdash^{(m+m',e+e',s+s'+1)} t\,u : \mathtt{n}}\,(\mathtt{app}_\mathtt{p}^{\mathcal{V}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \tau \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\Gamma(x))}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x\backslash u] : \tau}\,(\mathtt{es_p})$$

Figure 3 System $\mathcal{V}$ for the Call-by-Value Calculus: Persistent Typing Rules.

$$\frac{}{x : \mathcal{M} \vdash^{(0,1,0)} x : \mathcal{M}}\,(\mathtt{var}_\mathtt{c}^{\mathcal{V}}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : [\mathcal{M} \to \tau] \quad \Delta \vdash^{(m',e',s')} u : \mathcal{M}}{\Gamma + \Delta \vdash^{(m+m'+1,e+e'-1,s+s')} t\,u : \tau}\,(\mathtt{app}_\mathtt{c}^{\mathcal{V}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : [\mathcal{M} \to \tau] \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\mathcal{M})}{\Gamma + \Delta \vdash^{(m+m'+1,e+e'-1,s+s')} t\,u : \tau}\,(\mathtt{appt}_\mathtt{c}^{\mathcal{V}})$$

$$\frac{(\Gamma_i \vdash^{(m_i,e_i,s_i)} t : \tau_i)_{i\in I}}{+_{i\in I}\,\Gamma_i \setminus\!\setminus x \vdash^{(+_{i\in I} m_i, 1+_{i\in I} e_i, +_{i\in I} s_i)} \lambda x.t : [\Gamma_i(x) \to \tau_i]_{i\in I}}\,(\mathtt{abs}_\mathtt{c}^{\mathcal{V}})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \sigma \quad \Delta \vdash^{(m',e',s')} u : \Gamma(x)}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x\backslash u] : \sigma}\,(\mathtt{es_c})$$

Figure 4 System $\mathcal{V}$ for the Call-by-Value Calculus: Consuming Typing Rules.

Some rules deserve a comment. A difficult property to be statically captured by the counters is that an exponential step can only be generated by the meeting of a substitution constructor with an appropriate value argument. This remark leads to the introduction of different rules for typing variables, depending on the role they play (to be a value or not). Indeed, there are three axioms for variables: $(\mathtt{var_p})$ typing variables that will persist as the head of a neutral term; $(\mathtt{val_p})$ typing variable values that may be substituted by other values, *i.e.* they are *placeholders* for future persistent values; and $(\mathtt{var}_\mathtt{c}^{\mathcal{V}})$ typing variable values that, in particular, may be consumed as arguments. Consuming values are always typed with multitypes. Rule $(\mathtt{app}_\mathtt{p}^{\mathcal{V}})$ types neutral applications, *i.e.* the left premise has type $\mathtt{vr}$ or $\mathtt{n}$. Rule $(\mathtt{abs}_\mathtt{c}^{\mathcal{V}})$ increases the second counter, just like $(\mathtt{var}_\mathtt{c}^{\mathcal{V}})$, typing a value that is consumed as an argument. Rules $(\mathtt{app}_\mathtt{c}^{\mathcal{V}})$ and $(\mathtt{appt}_\mathtt{c}^{\mathcal{V}})$ increment the first counter because the (consuming) application will be used to perform a $\mathtt{dB}$-step, while they decrement the second counter to compensate for the left-hand-side value that is not being consumed after all. In other words, consuming values are systematically typed by incrementing their exponential counter by one (rules $(\mathtt{var}_\mathtt{c}^{\mathcal{V}})$ and $(\mathtt{abs}_\mathtt{c}^{\mathcal{V}})$), but they can finally act as computations instead as values, notably when they are placed in a head position, so that their exponential counter needs to be adjusted correctly (*cf.* Ex. 11). The decrement of the exponential counters in rules $(\mathtt{app}_\mathtt{c}^{\mathcal{V}})$ and $(\mathtt{appt}_\mathtt{c}^{\mathcal{V}})$ can also be understood by means of the subtle translation from CBV to the $\lambda!$-calculus that we introduce in Sec. 7. Rule $(\mathtt{appt}_\mathtt{c}^{\mathcal{V}})$ is particularly useful to type $\mathtt{dB}$-redexes whose reduction does not create an exponential redex, because the argument of the substitution created by the $\mathtt{dB}$-step does not reduce to a value.

In spite of the decrements in rules $(\mathtt{app}_\mathtt{c}^{\mathcal{V}})$ and $(\mathtt{appt}_\mathtt{c}^{\mathcal{V}})$, the counters are positive.

▶ **Lemma 4.** *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *then* $e \geq 0$*. Moreover, if* σ *is a multitype then* $e > 0$*.*

Soundness. The soundness property is based on a series of auxiliary results that enables to reason about tight type derivations, we only state here the more important ones.

▶ **Lemma 5** (Tight Spreading). *Let* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *such that* Γ *is tight.*
1. *If* $t \in \mathsf{ne}_{\mathtt{v}}$ *or* $m = e = 0$*, then* $\sigma \in \mathtt{tt}$*.*
2. *If* $\sigma = [\mathcal{M} \to \tau]$*, then* $t \notin \mathsf{vr}_{\mathtt{v}}$*.*

▶ **Lemma 6** (Exact Subject Reduction). *Let* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *be a tight derivation. If* $t \to_{\mathtt{dv}} t'$*, then there is* $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash^{(m',e',s)} t' : \sigma$ *such that*
(1) $m' = m - 1$ *and* $e' = e$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{m}$*-step;*
(2) $e' = e - 1$ *and* $m' = m$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{e}$*-step.*

An interesting remark is that types of subterms in tight derivations may change during CBV reduction, unlike other approaches [1, 40]. To illustrate this phenomenon, consider the following reduction $x[x\backslash \mathtt{I}] \to_{\mathtt{dv}} \mathtt{I}$. The terms on the left and right hand side can be resp. tightly typed by the following derivations (we omit the counters):

$$\dfrac{\dfrac{}{\vdash x : \mathtt{vl}}\,(\mathtt{val_p}) \qquad \dfrac{}{\vdash \mathtt{I} : [\,]}\,(\mathtt{abs}_{\mathtt{c}}^{\mathcal{V}})}{\vdash x[x\backslash \mathtt{I}] : \mathtt{vl}}\,(\mathtt{es_c}) \qquad\qquad \dfrac{}{\vdash \mathtt{I} : \mathtt{vl}}\,(\mathtt{abs}_{\mathtt{p}}^{\mathcal{V}})$$

Notice that the identity function $\mathtt{I}$ is typed differently in each derivation: the substitution introduced by rule $(\mathtt{es_c})$ is a consuming constructor, which disappears when $\mathtt{sv}$-reduction consumes its value argument, a phenomenon that is captured by means of a multitype for the argument of the substitution. Indeed, $\mathtt{I}$ on the left-hand side derivation is typed with the multitype $[\,]$. This value $\mathtt{I}$ substitutes a variable x typed with $\mathtt{vl}$, which is just a placeholder for a persistent value. Thus, once the substitution is performed, the identity $\mathtt{I}$ becomes persistent on the right-hand side, a phenomenon which is naturally captured by the tight type $\mathtt{vl}$. This observation also applies to the tight typing system of CBPV in Sec. 6.

▶ **Theorem 7** (Soundness). *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight, then there exists* p *such that* $p \in \mathsf{no}_{\mathtt{v}}$ *and* $t \twoheadrightarrow_{\mathtt{v}}^{(m,e)} p$ *with* m $\mathtt{m}$*-steps,* e $\mathtt{e}$*-steps, and* $|p|_{\mathtt{v}} = s$*.*

As in CBN, the previous theorem is stated by using the general notion of reduction $\to_{\mathtt{v}}$, but the proofs (notably Lem. 6) are done using the deterministic reduction $\to_{\mathtt{dv}}$. Similar comment applies to the forthcoming Thm. 10.

Completeness. The completeness result is also based on intermediate lemmas, we only state here the so-called *tight typing of normal forms* and *subject expansion* properties.

▶ **Lemma 8** (Tight Typing of Normal Forms). *If* $t \in \mathsf{no}_{\mathtt{v}}$*, then there is a tight derivation* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(0,0,|t|_{\mathtt{v}})} t : \sigma$*.*

▶ **Lemma 9** (Exact Subject Expansion). *Let* $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash^{(m',e',s)} t' : \sigma$ *be a tight derivation. If* $t \to_{\mathtt{dv}} t'$*, then there is* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *such that*
(1) $m' = m - 1$ *and* $e' = e$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{m}$*-step;*
(2) $e' = e - 1$ *and* $m' = m$ *if* $t \to_{\mathtt{dv}} t'$ *is an* $\mathtt{e}$*-step.*

Notice that tight derivations properly counts, separately, $\mathtt{m}$-steps and $\mathtt{e}$-steps. As a consequence, completeness follows.

▶ **Theorem 10** (Completeness). *If* $t \twoheadrightarrow_{\mathsf{v}}^{(m,e)} p$ *with* $p \in \mathsf{no}_{\mathsf{v}}$*, then there exists a tight type derivation* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,|p|_{\mathsf{v}})} t : \sigma$*.*

▶ **Example 11.** Consider $t_0 = \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})$ from Ex. 3 but now in the CBV setting. It $\to_{\mathsf{v}}$-reduces in 3 $\mathtt{m}$-steps and 2 $\mathtt{e}$-steps to $x[x\backslash z\,\mathtt{I}] \in \mathsf{no}_{\mathsf{v}}$, whose $\mathtt{v}$-size is 1, as follows:

$$\begin{array}{lllllll} t_0 & = & \underline{\mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})} & \to_{\mathtt{dB}} & \underline{(\lambda y.x)[x\backslash z\,\mathtt{I}]\,(\mathtt{I}\,\mathtt{I})} & \to_{\mathtt{dB}} & x[y\backslash \underline{\mathtt{I}\,\mathtt{I}}][x\backslash z\,\mathtt{I}] \\ & \to_{\mathtt{dB}} & \underline{x[y\backslash w[w\backslash \mathtt{I}]}][x\backslash z\,\mathtt{I}] & \to_{\mathtt{sv}} & \underline{x[w\backslash \mathtt{I}]}[x\backslash z\,\mathtt{I}] & \to_{\mathtt{sv}} & x[x\backslash z\,\mathtt{I}] \end{array}$$

The reader may check that system $\mathcal{V}$ indeed admits a proper tight type derivation for t_0 with final counters $(3, 2, 1)$, as expected.

5 The λ!-Calculus

This section briefly presents the ***bang calculus at a distance*** [12], called λ!-calculus. It is a (conservative) extension of the original bang calculus [25, 26], it uses ES operators and *reduction at a distance* [5], thus integrating commutative conversions without jeopardising confluence (see [12] for a discussion). Indeed, T. Ehrhard [25] studies the CBPV from a Linear Logic (LL) point of view by extending the λ-calculus with two new unary constructors *bang* (!) and *dereliction* (der), playing the role of the CBPV primitives `thunk`/`force` respectively. His calculus suffers from the absence of *commutative conversions* [45, 15], making some redexes to be syntactically blocked when open terms are considered. As a consequence, some normal forms are semantically equivalent to non-terminating programs, a situation which is clearly unsound. The bang calculus [26] adds commutative conversions specified by means of σ-reduction rules, which are crucial to unveil hidden (blocked) redexes. This approach, however, presents a major drawback since the resulting combined reduction relation is not confluent. The λ!-calculus [12] fixes these two problems at the same time. Indeed, the syntax of the bang calculus is enriched with explicit substitutions (ES), and σ-equivalence is integrated in the primary reduction system by using the *distance* paradigm [5], without any need to unveil hidden redexes by means of an independent relation.

We consider the following grammar for terms (denoted by $\mathcal{T}$) and contexts:

$$\begin{array}{rrcl} \textbf{\textit{(Terms)}} & t, u & ::= & x \in \mathcal{X} \mid t\,u \mid \lambda x.t \mid !\,t \mid \mathrm{der}\,t \mid t[x\backslash u] \\ \textbf{\textit{(List contexts)}} & \mathtt{L} & ::= & \Box \mid \mathtt{L}[x\backslash t] \\ \textbf{\textit{(Surface contexts)}} & \mathtt{S} & ::= & \Box \mid \mathtt{S}\,t \mid t\,\mathtt{S} \mid \lambda x.\mathtt{S} \mid \mathrm{der}\,\mathtt{S} \mid \mathtt{S}[x\backslash u] \mid t[x\backslash \mathtt{S}] \end{array}$$

Special terms are $\Delta_! = \lambda x.x\,!\,x$, and $\Omega_! = \Delta_!\,!\,\Delta_!$. Surface contexts do not allow the symbol $\Box$ to occur inside the bang constructor $!$. This is similar to *weak* contexts in λ-calculus, where $\Box$ cannot occur inside λ-abstractions. As we will see in Sec. 7, surface reduction in the λ!-calculus is perfectly sufficient to capture head reduction in CBN, disallowing reduction inside arguments, as well as open CBV, disallowing reduction inside abstractions. Finally, we define the ***f-size*** of terms as follows:

$$\begin{array}{llllllllll} |x|_{\mathtt{f}} & := & 0 & |\mathrm{der}\,t|_{\mathtt{f}} & := & |t|_{\mathtt{f}} & |t[x\backslash u]|_{\mathtt{f}} & := & |t|_{\mathtt{f}} + |u|_{\mathtt{f}} \\ |!\,t|_{\mathtt{f}} & := & 0 & |\lambda x.t|_{\mathtt{f}} & := & 1 + |t|_{\mathtt{f}} & |t\,u|_{\mathtt{f}} & := & 1 + |t|_{\mathtt{f}} + |u|_{\mathtt{f}} \end{array}$$

The ***λ!-calculus*** is given by the set of terms $\mathcal{T}$ and the ***(surface) reduction relation*** $\to_{\mathtt{f}}$, which is defined as the *union* of $\to_{\mathtt{dB}}$, $\to_{\mathtt{s!}}$ (**s**ubstitute bang) and $\to_{\mathtt{d!}}$ (**d**istant bang), defined respectively as the closure by contexts $\mathtt{S}$ of the following three rewriting rules:

$$\begin{array}{lll} \mathtt{L}\langle\lambda x.t\rangle\,u & \mapsto_{\mathtt{dB}} & \mathtt{L}\langle t[x\backslash u]\rangle \\ t[x\backslash \mathtt{L}\langle !\,u\rangle] & \mapsto_{\mathtt{s!}} & \mathtt{L}\langle t\,\{x\backslash u\}\rangle \\ \mathrm{der}\,(\mathtt{L}\langle !\,t\rangle) & \mapsto_{\mathtt{d!}} & \mathtt{L}\langle t\rangle \end{array}$$

The rules are defined *at a distance*, as in CBN/CBV, in the sense that the list context $\mathtt{L}$ allows the main constructors involved in the rules to be separated by an arbitrary finite list of substitutions. This new formulation integrates commutative conversions inside the main (logical) reduction rules of the calculus, thus inheriting the benefits enumerated in Sec. 1. Indeed, rule $\mathtt{s}!$ can be decomposed in two different rules $t[x\backslash !\, u] \mapsto t\,\{x\backslash u\}$ and $t[x\backslash \mathtt{L}\langle !\, u\rangle] \mapsto \mathtt{L}\langle t[x\backslash !\, u]\rangle$, while $\mathtt{d}!$ can be decomposed in $\mathrm{der}\,(!\, t) \mapsto t$ and $\mathrm{der}\,(\mathtt{L}\langle !\, t\rangle) \mapsto \mathtt{L}\langle \mathrm{der}\, !\, t\rangle$. We write $\twoheadrightarrow_{\mathtt{f}}$ for the reflexive-transitive closure of $\to_{\mathtt{f}}$. Given the translation of the bang calculus into LL proof-nets [25], we refer to $\mathtt{dB}$-steps as $\mathtt{m}$***ultiplicative*** and $\mathtt{s}!/\mathtt{d}!$-steps as $\mathtt{e}$***xponential*** steps. We write $t \twoheadrightarrow_{\mathtt{f}}^{(m,e)} u$ if $t \twoheadrightarrow_{\mathtt{f}} u$ using m $\mathtt{m}$-steps and e $\mathtt{e}$-steps.

▶ **Example 12.** Consider the following reduction sequence from $t'_0 = \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I}))$:

$$t'_0 = \underline{\mathtt{K}\,(!\,(z\,!\,\mathtt{I}))}\,(!\,(\mathtt{I}\,!\,\mathtt{I})) \quad \begin{array}{l}\to_{\mathtt{dB}} \\ \to_{\mathtt{s}!}\end{array} \quad \begin{array}{l}\underline{(\lambda y.x)[x\backslash !\,(z\,!\,\mathtt{I})]\,(!\,(\mathtt{I}\,!\,\mathtt{I}))} \\ \underline{(z\,!\,\mathtt{I})[y\backslash !\,(\mathtt{I}\,!\,\mathtt{I})]}\end{array} \quad \begin{array}{l}\to_{\mathtt{dB}} \\ \to_{\mathtt{s}!}\end{array} \quad \begin{array}{l}\underline{x[y\backslash !\,(\mathtt{I}\,!\,\mathtt{I})][x\backslash !\,(z\,!\,\mathtt{I})]} \\ z\,!\,\mathtt{I}\end{array}$$

Notice that the second $\mathtt{dB}$-step uses action at a distance, where $\mathtt{L}$ is $\Box[x\backslash !\,(z\,!\,\mathtt{I})]$.

The relation $\to_{\mathtt{f}}$ enjoys a weak diamond property, *i.e.* one-step divergence can be closed in one step if the diverging terms are different. This property has two important consequences.

▶ **Theorem 13** (Confluence [12])**.** *The reduction relation $\to_{\mathtt{f}}$ is confluent. Moreover, any two different $\to_{\mathtt{f}}$-reduction paths to normal form have the same length.*

The second point relies essentially on the fact that reductions are disallowed under bangs. An important consequence is that we can focus on any particular *deterministic* strategy for the $\lambda!$-calculus, without changing the number of steps to $\mathtt{f}$-normal form.

Neutral, Normal, and Clash-Free Terms. A term is said to be $\mathtt{f}$***-normal*** if there is no t' such that $t \to_{\mathtt{f}} t'$, in which case we write $t \not\to_{\mathtt{f}}$. However, some ill-formed $\mathtt{f}$-normal terms are not still the ones that represent a desired result for a computation, they are called ***clashes*** (meta-variable c), and take one of the following forms: $\mathtt{L}\langle !\, t\rangle\, u$, $t[y\backslash \mathtt{L}\langle \lambda x.u\rangle]$, $\mathrm{der}\,(\mathtt{L}\langle \lambda x.u\rangle)$, or $t\,(\mathtt{L}\langle \lambda x.u\rangle)$. Remark that in the three first kind of clashes, replacing $\lambda x.$ by !, and inversely, creates a (root) redex, namely $(\mathtt{L}\langle \lambda x.t\rangle)\, u$, $t[x\backslash \mathtt{L}\langle !\, t\rangle]$ and $\mathrm{der}\,(\mathtt{L}\langle !\, t\rangle)$, respectively.

A term is ***clash free*** if it does not reduce to a term containing a clash, it is ***surface clash free***, written $\mathtt{scf}$, if it does not reduce to a term containing a clash outside the scope of any constructor !. Thus, t is not $\mathtt{scf}$ if and only if there exist a surface context $\mathtt{S}$ and a clash c such that $t \twoheadrightarrow_{\mathtt{f}} \mathtt{S}\langle c\rangle$. Surface clash free normal terms can be characterised as follows:

$$\begin{array}{rlll} \textit{(Neutral } \mathtt{scf}\textit{)} & \mathsf{ne}_{\mathtt{scf}} & ::= & x \in \mathcal{X} \mid \mathsf{ne}_{\mathtt{scf}}\,\mathsf{na}_{\mathtt{scf}} \mid \mathrm{der}\,(\mathsf{ne}_{\mathtt{scf}}) \mid \mathsf{ne}_{\mathtt{scf}}[x\backslash \mathsf{ne}_{\mathtt{scf}}] \\ \textit{(Neutral-Abs } \mathtt{scf}\textit{)} & \mathsf{na}_{\mathtt{scf}} & ::= & !\, t \mid \mathsf{ne}_{\mathtt{scf}} \mid \mathsf{na}_{\mathtt{scf}}[x\backslash \mathsf{ne}_{\mathtt{scf}}] \\ \textit{(Neutral-Bang } \mathtt{scf}\textit{)} & \mathsf{nb}_{\mathtt{scf}} & ::= & \mathsf{ne}_{\mathtt{scf}} \mid \lambda x.\mathsf{no}_{\mathtt{scf}} \mid \mathsf{nb}_{\mathtt{scf}}[x\backslash \mathsf{ne}_{\mathtt{scf}}] \\ \textit{(Normal } \mathtt{scf}\textit{)} & \mathsf{no}_{\mathtt{scf}} & ::= & \mathsf{na}_{\mathtt{scf}} \mid \mathsf{nb}_{\mathtt{scf}} \end{array}$$

▶ **Proposition 14** (Clash-Free Normal Terms [12])**.** *Let $t \in \mathcal{T}$. Then t is a surface clash free $\mathtt{f}$-normal term iff $t \in \mathsf{no}_{\mathtt{scf}}$.*

6 A Tight Type System for the λ!-Calculus

The methodology used to define the type system $\mathcal{B}$ for the $\lambda!$-calculus is based on [12], inspired in turn from [19, 9, 1], which defines non-idempotent intersection type systems to count reduction lengths for different evaluation strategies in the λ-calculus. In the case of the

λ!-calculus, however, Thm. 13 guarantees that all reduction paths to normal form have the same length, so that it is not necessary to reason w.r.t. any particular evaluation strategy.

The grammar of types of system $\mathcal{B}$ is given by:

$$\begin{array}{rrcl} \textbf{\textit{(Tight Types)}} & \mathtt{tt} & ::= & \mathtt{n} \mid \mathtt{a} \mid \mathtt{vl} \\ \textbf{\textit{(Types)}} & \sigma, \tau & ::= & \mathtt{tt} \mid \mathcal{M} \mid \mathcal{M} \to \sigma \\ \textbf{\textit{(Multitype)}} & \mathcal{M}, \mathcal{N} & ::= & [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set} \end{array}$$

The constant $\mathtt{a}$ (resp. $\mathtt{vl}$) types terms whose normal form has the shape $\mathtt{L}\langle \lambda x.t \rangle$ (resp. $\mathtt{L}\langle !\, t \rangle$, *i.e. values* in the λ!-calculus sense), and the constant $\mathtt{n}$ types terms whose normal form is a neutral $\mathtt{scf}$. As before, the notions of ***tightness*** for multitypes, contexts and derivations are exactly the same used for CBN. Typing rules are split in two groups: the *persistent* rules (Fig. 5) and the *consuming* ones (Fig. 6).

$$\frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{n} \quad \Delta \vdash^{(m',e',s')} u : \overline{\mathtt{a}}}{\Gamma + \Delta \vdash^{(m+m',e+e',s+s'+1)} t\, u : \mathtt{n}}\,(\mathtt{app_p}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{tt} \quad \mathsf{tight}(\Gamma(x))}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s+1)} \lambda x.t : \mathtt{a}}\,(\mathtt{abs_p})$$

$$\frac{}{\vdash^{(0,0,0)} !\, t : \mathtt{vl}}\,(\mathtt{bg_p}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathtt{n}}{\Gamma \vdash^{(m,e,s)} \mathrm{der}\, t : \mathtt{n}}\,(\mathtt{dr_p}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \tau \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\Gamma(x))}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x \backslash u] : \tau}\,(\mathtt{es_p})$$

Figure 5 System $\mathcal{B}$ for the λ!-Calculus: Persistent Typing Rules.

$$\frac{}{x : [\sigma] \vdash^{(0,0,0)} x : \sigma}\,(\mathtt{var_c}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \mathcal{M} \to \tau \quad \Delta \vdash^{(m',e',s')} u : \mathcal{M}}{\Gamma + \Delta \vdash^{(m+m'+1,e+e',s+s')} t\, u : \tau}\,(\mathtt{app_c})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \mathcal{M} \to \tau \quad \Delta \vdash^{(m',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\mathcal{M})}{\Gamma + \Delta \vdash^{(m+m'+1,e+e',s+s')} t\, u : \tau}\,(\mathtt{appt_c})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : \tau}{\Gamma \setminus\!\setminus x \vdash^{(m,e,s)} \lambda x.t : \Gamma(x) \to \tau}\,(\mathtt{abs_c}) \qquad \frac{(\Gamma_i \vdash^{(m_i,e_i,s_i)} t : \sigma_i)_{i \in I}}{+_{i \in I}\, \Gamma_i \vdash^{(+_{i\in I} m_i, 1 +_{i\in I} e_i, +_{i\in I} s_i)} !\, t : [\sigma_i]_{i \in I}}\,(\mathtt{bg_c})$$

$$\frac{\Gamma \vdash^{(m,e,s)} t : [\sigma]}{\Gamma \vdash^{(m,e,s)} \mathrm{der}\, t : \sigma}\,(\mathtt{dr_c}) \qquad \frac{\Gamma \vdash^{(m,e,s)} t : \sigma \quad \Delta \vdash^{(m',e',s')} u : \Gamma(x)}{(\Gamma \setminus\!\setminus x) + \Delta \vdash^{(m+m',e+e',s+s')} t[x \backslash u] : \sigma}\,(\mathtt{es_c})$$

Figure 6 System $\mathcal{B}$ for the λ!-Calculus: Consuming Typing Rules.

As in CBV, the $\mathtt{s}!$ exponential steps do not only depend on a (consuming) substitution constructor, but on the (bang) form of its argument. This makes the exponential counting more subtle. Notice also that rule $(\mathtt{dr_p})$ does not count der constructors, according to the definition of $|\_|_{\mathtt{f}}$ given in Sec. 5 and in contrast to [12]. This is to keep a more intuitive relation with the CBN/CBV translations (Sec. 2), where der plays a silent role.

As in [12], system $\mathcal{B}$ is quantitatively sound and complete. More precisely,

▶ **Theorem 15** (Soundness and Completeness).

1. *If* $\Phi \rhd_{\mathcal{B}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight, then there exists* p *such that* $p \in \mathsf{no}_{\mathtt{scf}}$ *and* $t \twoheadrightarrow_{\mathtt{f}}^{(m,e)} p$ *with* m $\mathtt{m}$*-steps,* e $\mathtt{e}$*-steps, and* $|p|_{\mathtt{f}} = s$.
2. *If* $t \twoheadrightarrow_{\mathtt{f}}^{(m,e)} p$ *with* $p \in \mathsf{no}_{\mathtt{scf}}$*, then there exists a tight type derivation* $\Phi \rhd_{\mathcal{B}} \Gamma \vdash^{(m,e,|p|_{\mathtt{f}})} t : \sigma$.

▶ **Example 16.** Consider $t'_0 = \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I}))$ from Ex. 12, which normalises in 2 $\mathtt{m}$-steps and 2 $\mathtt{e}$-steps to $z\,!\,\mathtt{I} \in \mathtt{no}_{\mathtt{scf}}$ of $\mathtt{f}$-size 1. A tight derivation for t'_0 with appropriate final counters $(2,2,1)$ is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{n}] \vdash^{(0,0,0)} x : \mathtt{n}}\,(\mathtt{var_c})}{x : [\mathtt{n}] \vdash^{(0,0,0)} \lambda y.x : [\,] \to \mathtt{n}}\,(\mathtt{abs_c})}{\vdash^{(0,0,0)} \mathtt{K} : [\mathtt{n}] \to [\,] \to \mathtt{n}}\,(\mathtt{abs_c}) \qquad \dfrac{\dfrac{\dfrac{}{z : [\mathtt{n}] \vdash^{(0,0,0)} z : \mathtt{n}}\,(\mathtt{var_c}) \quad \dfrac{}{\vdash^{(0,0,0)} !\,\mathtt{I} : \mathtt{vl}}\,(\mathtt{bg_p})}{z : [\mathtt{n}] \vdash^{(0,0,1)} z\,!\,\mathtt{I} : \mathtt{n}}\,(\mathtt{app_p})}{z : [\mathtt{n}] \vdash^{(0,1,1)} !\,(z\,!\,\mathtt{I}) : [\mathtt{n}]}\,(\mathtt{bg_c})}{z : [\mathtt{n}] \vdash^{(1,1,1)} \mathtt{K}\,(!\,(z\,!\,\mathtt{I})) : [\,] \to \mathtt{n}}\,(\mathtt{app_c}) \qquad \dfrac{}{\vdash^{(0,1,0)} !\,(\mathtt{I}\,!\,\mathtt{I}) : [\,]}\,(\mathtt{bg_c})}{z : [\mathtt{n}] \vdash^{(2,2,1)} \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I})) : \mathtt{n}}\,(\mathtt{app_c})$$

Notice that the only persistent rules are $(\mathtt{bg_p})$ and $(\mathtt{app_p})$, used to type $z\,!\,\mathtt{I}$. Indeed, $z\,!\,\mathtt{I}$ is the $\mathtt{f}$-normal form of t'_0.

7 Untyped Translations

CBN/CBV (untyped) encodings into the bang calculus [32], inspired from Girard's encodings, establish two translations *cbn* and *cbv*, such that when t reduces to u in CBN (resp. CBV), $\mathit{cbn}(t)$ reduces to $\mathit{cbn}(u)$ (resp. $\mathit{cbv}(t)$ reduces to $\mathit{cbv}(u)$) in the bang calculus. These two encodings are dual: CBN forbids reduction inside arguments, which are translated to bang terms, while CBV forbids reduction under λ-abstractions, also translated to bang terms.

In this paper we use alternative encodings. For CBN, we slightly adapt to explicit substitutions Girard's translation into LL [29]. The resulting encoding preserves normal forms and is sound and complete with respect to the standard (quantitative) type system in [28]. For CBV, we discard the original encoding in [32] for two reasons: CBV normal forms are not necessarily translated to normal forms in the bang calculus (see [32]), and levels of terms (the level of t is the number of ! surrounding t) are not preserved either (see [27]). We thus adopt the CBV encoding in [12] which preserves normal forms as well as levels.

The CBN and CBV embedding into the λ!-calculus, written $\_^{\mathtt{n}}$ and $\_^{\mathtt{v}}$ resp., are inductively defined as:

$$\begin{array}{rcl} x^{\mathtt{n}} & \stackrel{def}{=} & x \\ (\lambda x.t)^{\mathtt{n}} & \stackrel{def}{=} & \lambda x.t^{\mathtt{n}} \\ (t\,u)^{\mathtt{n}} & \stackrel{def}{=} & t^{\mathtt{n}}\,!\,u^{\mathtt{n}} \\ (t[x\backslash u])^{\mathtt{n}} & \stackrel{def}{=} & t^{\mathtt{n}}[x\backslash !\,u^{\mathtt{n}}] \end{array} \qquad \begin{array}{rcl} x^{\mathtt{v}} & \stackrel{def}{=} & !\,x \\ (\lambda x.t)^{\mathtt{v}} & \stackrel{def}{=} & !\,\lambda x.t^{\mathtt{v}} \\ (t\,u)^{\mathtt{v}} & \stackrel{def}{=} & \begin{cases} \mathtt{L}\langle s\rangle\,u^{\mathtt{v}} & \text{if } t^{\mathtt{v}} = \mathtt{L}\langle !\,s\rangle \\ \mathrm{der}\,(t^{\mathtt{v}})\,u^{\mathtt{v}} & \text{otherwise} \end{cases} \\ (t[x\backslash u])^{\mathtt{v}} & \stackrel{def}{=} & t^{\mathtt{v}}[x\backslash u^{\mathtt{v}}] \end{array}$$

Both translations extend to list contexts $\mathtt{L}$ as expected. Remark that there are no two consecutive ! constructors in the image of the translation. The CBN embedding extends Girard's translation to ES, while the CBV one is different. Indeed, the translation of an application $t\,u$ is usually defined as $\mathrm{der}\,(t^{\mathtt{v}})\,u^{\mathtt{v}}$ (see *e.g.* [26]). This definition does not preserve normal forms, *i.e.* $x\,y$ is a $\mathtt{v}$-normal form but its translated version $\mathrm{der}\,(!\,x)\,!\,y$ is not a $\mathtt{f}$-normal form. We restore this fundamental property by using the well-known notion of superdevelopment [10], so that $\mathtt{d}$!-reductions are applied by the translation on the fly. Moreover, simulation of CBN/CBV in the λ!-calculus also holds.

▶ **Lemma 17** (Simulation [12])**.** *Let $t \in \mathcal{T}_\lambda$.*

1. *$t \not\to_{\mathtt{n}}$ implies $t^{\mathtt{n}} \not\to_{\mathtt{f}}$, and $t \to_{\mathtt{n}} s$ implies $t^{\mathtt{n}} \to_{\mathtt{f}} s^{\mathtt{n}}$.*
2. *$t \not\to_{\mathtt{v}}$ implies $t^{\mathtt{v}} \not\to_{\mathtt{f}}$, and $t \to_{\mathtt{v}} s$ implies $t^{\mathtt{v}} \twoheadrightarrow_{\mathtt{f}} s^{\mathtt{v}}$.*

▶ **Example 18.** Consider again $t_0 = \mathtt{K}\,(z\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I})$. To illustrate the CBN case (Lem. 17:1), notice that the reduction sequence $t_0 \twoheadrightarrow_{\mathtt{cbn}} z\,\mathtt{I} = s_0$ given in Ex. 3 is translated to the sequence ${t_0}^{\mathtt{n}} = \mathtt{K}\,(!\,(z\,!\,\mathtt{I}))\,(!\,(\mathtt{I}\,!\,\mathtt{I})) = t'_0 \twoheadrightarrow_{\mathtt{f}} z\,!\,\mathtt{I} = {s_0}^{\mathtt{n}}$ given in Ex. 12.

To illustrate the CBV case (Lem. 17:2), consider the sequence $t_0 \twoheadrightarrow_{\mathtt{cbv}} x[x\backslash z\,\mathtt{I}] = s_1$ in Ex. 11. Then for $\mathtt{I}' = \lambda w.!\,w$ we have:

$$\begin{array}{lll}
{t_0}^{\mathtt{v}} = & \mathtt{der}\,(\underline{(\lambda x.!\,\lambda y.!\,x)\,(z\,!\,\mathtt{I}')})\,(\mathtt{I}'\,!\,\mathtt{I}') & \to_{\mathtt{dB}} \underline{\mathtt{der}\,((!\,\lambda y.!\,x)[x\backslash z\,!\,\mathtt{I}'])\,(\mathtt{I}'\,!\,\mathtt{I}')} \\
\to_{\mathtt{d!}} \underline{(\lambda y.!\,x)[x\backslash z\,!\,\mathtt{I}']\,(\mathtt{I}'\,!\,\mathtt{I}')} & & \to_{\mathtt{dB}} (!\,x)[y\backslash\underline{\mathtt{I}'\,!\,\mathtt{I}'}][x\backslash z\,!\,\mathtt{I}'] \\
\to_{\mathtt{dB}} \underline{(!\,x)[y\backslash(!\,w)[w\backslash!\,\mathtt{I}']]}[x\backslash z\,!\,\mathtt{I}'] & & \to_{\mathtt{s!}} \underline{(!\,x)[w\backslash!\,\mathtt{I}']}[x\backslash z\,!\,\mathtt{I}'] \\
\to_{\mathtt{s!}} (!\,x)[x\backslash z\,!\,\mathtt{I}'] & & = {s_1}^{\mathtt{v}}
\end{array}$$

Notice how this sequence requires extra reduction steps with respect to the one given in Ex. 11. Indeed, the $\mathtt{e}$-step $\to_{\mathtt{d!}}$ in the $\lambda!$-calculus has no counterpart in CBV.

8 Typed Translations

Call-by-Name. We study the correspondence between derivations in CBN and their encodings in the $\lambda!$-calculus. First we inject the set of types for $\mathcal{N}$ (generated by the base types $\mathtt{n}$ and $\mathtt{a}$) into the set of types of $\mathcal{B}$ (generated also by the base type $\mathtt{vl}$) by means of the function: $\mathtt{n}^{\mathtt{n}} \stackrel{def}{=} \mathtt{n}$, $\mathtt{a}^{\mathtt{n}} \stackrel{def}{=} \mathtt{a}$, $(\mathcal{M} \to \sigma)^{\mathtt{n}} \stackrel{def}{=} \mathcal{M}^{\mathtt{n}} \to \sigma^{\mathtt{n}}$ and $[\sigma_i]_{i\in I}{}^{\mathtt{n}} \stackrel{def}{=} [\sigma_i{}^{\mathtt{n}}]_{i\in I}$. Then we translate terms, using the function $\_^{\mathtt{n}}$ from Sec. 7. Translation of contexts is defined as expected: $\Gamma^{\mathtt{n}} = \{x_i : \mathcal{M}_i{}^{\mathtt{n}}\}_{i\in I}$. Another notion is needed to restrict $\mathcal{B}$ derivations to those that come from the translation of some $\mathcal{N}$ derivation. Indeed, a $\mathcal{B}$ derivation Φ is ***$\mathtt{n}$-relevant*** if all the contexts and types involved in Φ are in the image of the translation $\_^{\mathtt{n}}$. We then obtain:

▶ **Theorem 19.** *$\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ if and only if $\Phi' \rhd_{\mathcal{B}} \Gamma^{\mathtt{n}} \vdash^{(m,e,s)} t^{\mathtt{n}} : \sigma^{\mathtt{n}}$ is $\mathtt{n}$-relevant.*

This result is illustrated by the tight type derivations in Ex. 3 and 16 for the terms t_0 and t'_0 resp. Moreover, tightness is preserved by the translation of contexts and types, hence:

▶ **Corollary 20.** *If $\Phi \rhd_{\mathcal{N}} \Gamma \vdash^{(m,e,s)} t : \sigma$ is tight, then there exists $p \in \mathsf{no}_{\mathtt{scf}}$ such that $t^{\mathtt{n}} \twoheadrightarrow_{\mathtt{f}}^{(m,e)} p$ with m $\mathtt{m}$-steps, e $\mathtt{e}$-steps, and $|p|_{\mathtt{f}} = s$. Conversely, if $\Phi' \rhd_{\mathcal{B}} \Gamma^{\mathtt{n}} \vdash^{(m,e,s)} t^{\mathtt{n}} : \sigma^{\mathtt{n}}$ is tight and $\mathtt{n}$-relevant, then there exists $p \in \mathsf{no}_{\mathtt{n}}$ such that $t \twoheadrightarrow_{\mathtt{n}}^{(m,e)} p$ with m $\mathtt{m}$-steps, e $\mathtt{e}$-steps, and $|p|_{\mathtt{n}} = s$.*

This result shows that not only from the tight type system $\mathcal{N}$ it is possible to extract exact measures for the image of the CBN in the $\lambda!$-calculus, but more interestingly, also that from the tight type system $\mathcal{B}$ for the $\lambda!$-calculus it is possible to extract exact measures for CBN. In this sense, the goal of encoding tight typing in a unified framework is achieved.

Call-by-Value. In contrast with the CBN case, the set of type for system $\mathcal{V}$ is not a subset of that for $\mathcal{B}$, and we need to properly translate types. To that end, we introduce two mutually dependent translations $\_^{\overline{\mathtt{v}}}$ and $\_^{\mathtt{v}}$:

$$\begin{array}{rcll}
\mathtt{tt}^{\overline{\mathtt{v}}} & \stackrel{def}{=} & \mathtt{tt} & \text{if } \mathtt{tt} \neq \mathtt{vr} \\
\mathtt{vr}^{\overline{\mathtt{v}}} & \stackrel{def}{=} & \mathtt{n} & \\
(\mathcal{M} \to \sigma)^{\overline{\mathtt{v}}} & \stackrel{def}{=} & \mathcal{M}^{\overline{\mathtt{v}}} \to \sigma^{\mathtt{v}} & \\
([\sigma]_{i\in I})^{\overline{\mathtt{v}}} & \stackrel{def}{=} & [\sigma_i{}^{\overline{\mathtt{v}}}]_{i\in I} &
\end{array}
\qquad
\begin{array}{rcll}
\mathtt{tt}^{\mathtt{v}} & \stackrel{def}{=} & \mathtt{tt} & \text{if } \mathtt{tt} \neq \mathtt{vr} \\
\mathtt{vr}^{\mathtt{v}} & \stackrel{def}{=} & [\mathtt{n}] & \\
(\mathcal{M} \to \sigma)^{\mathtt{v}} & \stackrel{def}{=} & \mathcal{M}^{\overline{\mathtt{v}}} \to \sigma^{\mathtt{v}} & \\
([\sigma]_{i\in I})^{\mathtt{v}} & \stackrel{def}{=} & [\sigma_i{}^{\overline{\mathtt{v}}}]_{i\in I} &
\end{array}$$

Remark that $\mathcal{M}^{\overline{\mathtt{v}}} = \mathcal{M}^{\mathtt{v}}$ for every multitype $\mathcal{M}$. Translation $\_^{\overline{\mathtt{v}}}$ for a context Γ is defined as expected: $\Gamma^{\overline{\mathtt{v}}} = \{x_i : \mathcal{M}_i{}^{\overline{\mathtt{v}}}\}_{i\in I}$. To translate terms, we resort to the function $\_^{\mathtt{v}}$ defined in Sec. 7. We also restrict $\mathcal{B}$ derivations to those that come from the translation of some $\mathcal{V}$ derivation. Indeed, a $\mathcal{B}$ derivation Φ is ***$\mathtt{v}$-relevant*** if:

(1) all the contexts and types involved in Φ are in the image of the translations $\_^{\overline{\mathtt{v}}}$ and $\_^{\mathtt{v}}$ respectively; and

(2) rule $(\mathtt{dr_c})$ is only applied to terms having a type of the form $[\mathcal{M} \to \tau]$.

To state the preservation of typing derivations between systems $\mathcal{V}$ and $\mathcal{B}$ we define measures over type derivations in both systems to conveniently capture the relationship between the exponential steps in the source and the target derivation. The intuition is that we need to compensate for those $\mathtt{d}!$-redexes of the $\lambda!$-calculus that might be introduced when translating. For all the typing rules with two premises, we write Φ_t and Φ_u for the first and second premise respectively. The first measure for system $\mathcal{V}$ is given by induction on Φ as follows:

(1) for $(\mathtt{var_p})$, $\mathtt{e}(\Phi) \stackrel{def}{=} 1$;

(2) for $(\mathtt{val_p})$, $(\mathtt{abs}^{\mathcal{V}}_{\mathtt{p}})$ and $(\mathtt{var}^{\mathcal{V}}_{\mathtt{c}})$, $\mathtt{e}(\Phi) \stackrel{def}{=} 0$;

(3) for $(\mathtt{app}^{\mathcal{V}}_{\mathtt{p}})$, $\mathtt{e}(\Phi) \stackrel{def}{=} \mathtt{e}(\Phi_t) + \mathtt{e}(\Phi_u) - 1$ if $\mathsf{val}(t)$;

(4) for $(\mathtt{app}^{\mathcal{V}}_{\mathtt{c}})$ and $(\mathtt{appt}^{\mathcal{V}}_{\mathtt{c}})$, $\mathtt{e}(\Phi) \stackrel{def}{=} \mathtt{e}(\Phi_t) + \mathtt{e}(\Phi_u) + 1$ if $\neg\mathsf{val}(t)$; and

(5) in any other case $\mathtt{e}(\Phi)$ is defined as the sum of the recursive calls over all premises.

The second measure for system $\mathcal{B}$ is also defined by induction on Φ as follows:

(1) for $(\mathtt{bg_p})$ and $(\mathtt{var_c})$, $\widehat{\mathtt{e}}(\Phi) \stackrel{def}{=} 0$;

(2) for $(\mathtt{app_p})$, $\widehat{\mathtt{e}}(\Phi) \stackrel{def}{=} \widehat{\mathtt{e}}(\Phi_t) + \widehat{\mathtt{e}}(\Phi_u) - 1$ if $\mathsf{val}(t)$;

(3) for $(\mathtt{app_c})$ and $(\mathtt{appt_c})$, $\widehat{\mathtt{e}}(\Phi) \stackrel{def}{=} \widehat{\mathtt{e}}(\Phi_t) + \widehat{\mathtt{e}}(\Phi_u) + 1$ if $\neg\mathsf{val}(t)$; and

(4) in any other case $\widehat{\mathtt{e}}(\Phi)$ is defined as the sum of the recursive calls over all premises.

Then, we obtain:

▶ **Theorem 21.**

1. *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$, *then* $\Phi' \rhd_{\mathcal{B}} \Gamma^{\overline{\mathtt{v}}} \vdash^{(m,e',s)} t^{\mathtt{v}} : \sigma^{\mathtt{v}}$ *is* $\mathtt{v}$*-relevant with* $e' = e + \mathtt{e}(\Phi)$.
2. *If* $\Phi' \rhd_{\mathcal{B}} \Gamma^{\overline{\mathtt{v}}} \vdash^{(m,e',s)} t^{\mathtt{v}} : \sigma^{\mathtt{v}}$ *is* $\mathtt{v}$*-relevant, then* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *with* $e = e' - \widehat{\mathtt{e}}(\Phi')$.

As an example, consider the CBV term $(\lambda x.x)\, y$ (whose derivation Φ is on the left) and its translation $(\lambda x.!\, x)\,!\, y$ into the $\lambda!$-calculus (whose derivation Φ' is on the right):

$$\dfrac{\dfrac{\dfrac{}{x : [\mathtt{vr}] \vdash^{(0,0,0)} x : \mathtt{vr}}}{\vdash^{(0,1,0)} \lambda x.x : [[\mathtt{vr}] \to \mathtt{vr}]} \quad \dfrac{}{y : [\mathtt{vr}] \vdash^{(0,1,0)} y : [\mathtt{vr}]}}{y : [\mathtt{vr}] \vdash^{(1,1,0)} (\lambda x.x)\, y : \mathtt{vr}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{n}] \vdash^{(0,0,0)} x : \mathtt{n}}}{x : [\mathtt{n}] \vdash^{(0,1,0)} !\, x : [\mathtt{n}]}}{\vdash^{(0,1,0)} \lambda x.!\, x : [\mathtt{n}] \to [\mathtt{n}]} \quad \dfrac{\dfrac{}{y : [\mathtt{n}] \vdash^{(0,0,0)} y : \mathtt{n}}}{y : [\mathtt{n}] \vdash^{(0,1,0)} !\, y : [\mathtt{n}]}}{y : [\mathtt{n}] \vdash^{(1,2,0)} (\lambda x.!\, x)\,!\, y : [\mathtt{n}]}$$

Notice that $\mathsf{tight}(\Gamma)$ if and only if $\mathsf{tight}(\Gamma^{\overline{\mathtt{v}}})$. Unfortunately, this is not sufficient to translate a tight derivation in $\mathcal{V}$ into a tight derivation in $\mathcal{B}$. Indeed, the variable x of type $\mathtt{vr}$ translates to $!\, x$ of type $[\mathtt{n}]$. However, this only happens if the derived type is $\mathtt{vr}$, which is an auxiliary type of system $\mathcal{V}$ used to identify variables that are not used as values because they will be applied to some argument. In the case of ***$\mathtt{f}$-relevant $\mathcal{V}$ derivations***, defined as not deriving type $\mathtt{vr}$, tightness is indeed preserved. As a consequence, as for CBN, is it possible to study CBV in the unified framework of the $\lambda!$-calculus and extract exact measures for it by resorting to relevant tight derivations:

▶ **Corollary 22.** *If* $\Phi \rhd_{\mathcal{V}} \Gamma \vdash^{(m,e,s)} t : \sigma$ *is tight and* $\mathtt{f}$*-relevant, then there exists* $p \in \mathsf{no}_{\mathtt{scf}}$ *such that* $t^{\mathtt{v}} \twoheadrightarrow^{(m,e)}_{\mathtt{f}} p$ *with* m $\mathtt{m}$*-steps,* $e + \mathtt{e}(\Phi)$ $\mathtt{e}$*-steps, and* $|p|_{\mathtt{f}} = s$. *Conversely, if* $\Phi' \rhd_{\mathcal{B}} \Gamma^{\overline{\mathtt{v}}} \vdash^{(m,e',s)} t^{\mathtt{v}} : \sigma^{\mathtt{v}}$ *is tight and* $\mathtt{v}$*-relevant, then there exists* $p \in \mathsf{no}_{\mathtt{v}}$ *such that* $t \twoheadrightarrow^{(m,e)}_{\mathtt{v}} p$ *with* m $\mathtt{m}$*-steps,* $e' - \widehat{\mathtt{e}}(\Phi')$ $\mathtt{e}$*-steps, and* $|p|_{\mathtt{v}} = s$.

9 Conclusion

Following recent works exploring the power of CBPV, we develop a technique for deriving tight type systems for CBN/CBV as special cases of a single tight type system for the $\lambda!$-calculus, a subcalculus of CBPV inspired by Linear Logic.

The idea to study semantical and operational properties in a CBPV framework in order to transfer them to CBN/CBV has so far be exploited in different works [42, 25, 26, 32, 16, 12, 46]. Moreover, relational models for CBN/CBV can be derived from the relational model for CBPV, resulting in non-idempotent intersection type systems for them, that provide upper bounds for the length of normalization sequences [12]. However, the challenging quest of a (tight) quantitative type system for CBV, giving *exact measures* for the length of normalization sequences instead of *upper bounds*, and being at the same time encodable in CBPV, has been open. None of the existing proposals [3, 40] could be defined/explained within such an approach. In particular, the tight type systems that we propose for CBN/CBV give independent exact measures for the length of multiplicative and exponential reduction to normal form, as well as the size of these normal forms.

Different topics deserve future attention. One of them is the study of *strong* reduction for the $\lambda!$-calculus, which allows to reduce terms under *all* the constructors, including bang. Appropriate encodings of strong CBN and strong CBV should follow. Linear (head) reduction, as well as other more sophisticated semantics like GOI also deserve some attention.The tight systems presented in this work could also be used to understand bounded computation.

References

1 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *PACMPL*, 2(ICFP):94:1–94:30, 2018. doi:10.1145/3236789.

2 Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. In Atsushi Igarashi, editor, *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. doi:10.1007/978-3-319-47958-3_12.

3 Beniamino Accattoli and Giulio Guerrieri. Types of fireballs. In Sukyoung Ryu, editor, *Programming Languages and Systems – 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, volume 11275 of *Lecture Notes in Computer Science*, pages 45–66. Springer, 2018. doi:10.1007/978-3-030-02768-1_3.

4 Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In Luís Caires, editor, *Programming Languages and Systems – 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 410–439. Springer, 2019. doi:10.1007/978-3-030-17184-1_15.

5 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

6 Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming – 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16. Springer, 2012. doi:10.1007/978-3-642-29822-6_4.

7 Sandra Alves, Delia Kesner, and Daniel Ventura. A quantitative understanding of pattern matching. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, volume 175 of *LIPIcs*, pages 3:1–3:36. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.TYPES.2019.3.

8 Hendrik P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*, volume 103. North Holland, revised edition, 1984.

9 Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:3)2013.

10 Marc Bezem, Jan Willem Klop, and Vincent van Oostrom. *Term Rewriting Systems (TeReSe)*. Cambridge University Press, 2003.

11 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Logic*, 109(3):205–241, 2001. doi:10.1016/S0168-0072(00)00056-7.

12 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. doi:10.1007/978-3-030-59025-3_2.

13 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Solvability = typability + inhabitation. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: https://lmcs.episciences.org/7141.

14 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017. doi:10.1093/jigpal/jzx018.

15 Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures – 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2014. doi:10.1007/978-3-642-54830-7_7.

16 Jules Chouquet and Christine Tasson. Taylor expansion for call-by-push-value. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CSL.2020.16.

17 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, pages 233–243. ACM, 2000. doi:10.1145/351240.351262.

18 Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Sequent calculi for second order logic. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*. Cambridge University Press, 1995.

19 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. PhD thesis, Université Aix-Marseille II, 2007.

20 Daniel de Carvalho. The relational model is injective for multiplicative exponential linear logic. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 – September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 41:1–41:19. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.41.

21 Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, 28(7):1169–1203, 2018. doi:10.1017/S0960129516000396.

22 Daniel de Carvalho and Lorenzo Tortora de Falco. A semantic account of strong normalization in linear logic. *Inf. Comput.*, 248:104–129, 2016. doi:10.1016/j.ic.2015.12.010.

23 Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. A semantic measure of the execution time in linear logic. *Theor. Comput. Sci.*, 412(20):1884–1902, 2011. doi:10.1016/j.tcs.2010.12.017.

24 Thomas Ehrhard. Collapsing non-idempotent intersection types. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) – 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.259.

25 Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In Peter Thiemann, editor, *Programming Languages and Systems – 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016. doi:10.1007/978-3-662-49498-1_9.

26 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. doi:10.1145/2967973.2968608.

27 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, *Foundations of Software Science and Computation Structures – 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021. doi:10.1007/978-3-030-71995-1_11.

28 Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 1994. doi:10.1007/3-540-57887-0_115.

29 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

30 Jean-Yves Girard. Normal functors, power series and λ-calculus. *Ann. Pure Appl. Logic*, 37(2):129–177, 1988. doi:10.1016/0168-0072(88)90025-5.

31 Giulio Guerrieri. Towards a semantic measure of the execution time in call-by-value lambda-calculus. In Michele Pagani and Sandra Alves, editors, *Proceedings Twelfth Workshop on Developments in Computational Models and Ninth Workshop on Intersection Types and Related Systems, DCM/ITRS 2018, Oxford, UK, 8th July 2018*, volume 293 of *EPTCS*, pages 57–72, 2018. doi:10.4204/EPTCS.293.5.

32 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), Oxford, UK, 7-8 July 2018.*, EPTCS, pages 15–30, 2019.

33 Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. Computing connected proof(-structure)s from their taylor expansion. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.20.

34 Axel Kerinec, Giulio Manzonetto, and Simona Ronchi Della Rocca. Call-by-value, again! In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.7.

35 Delia Kesner and Pierre Vial. Types as resources for classical natural deduction. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 24:1–24:17. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.24.

36 Delia Kesner and Pierre Vial. Consuming and persistent types for classical logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 619–632. ACM, 2020. doi:10.1145/3373718.3394774.

37 Delia Kesner and Andrés Viso. Encoding tight typing in a unified framework. *CoRR*, abs/2105.00564, 2021. arXiv:2105.00564.

38 Zurab Khasidashvili. The Church-Rosser theorem in orthogonal combinatory reduction systems. Technical Report 1825, INRIA Rocquencourt, France, 1992.

39 Jan Willem Klop. *Combinatory reduction systems*. PhD thesis, Univ. Utrecht, 1980.

40 Maico Leberle. *Dissecting call-by-need by customizing multi type systems*. PhD thesis, Institut Polytechnique de Paris, 2021.

41 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.

42 Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006. doi:10.1007/s10990-006-0480-6.

43 Giulio Manzonetto, Michele Pagani, and Simona Ronchi Della Rocca. New semantical insights into call-by-value λ-calculus. *Fundam. Informaticae*, 170(1-3):241–265, 2019. doi:10.3233/FI-2019-1862.

44 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *Proc. ACM Program. Lang.*, 2(POPL):6:1–6:28, 2018. doi:10.1145/3158094.

45 Laurent Regnier. Une équivalence sur les lambda-termes. *TCS*, 2(126):281–292, 1994.

46 José Espírito Santo, Luís Pinto, and Tarmo 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. doi:10.4230/LIPIcs.FSCD.2019.18.