The Bang Calculus Revisited

Antonio Bucciarelli[1], Delia Kesner[1,2], Alejandro Ríos[3], and Andrés Viso[3,4]

[1] IRIF, CNRS and Université de Paris, France
[2] Institut Universitaire de France, France
[3] Universidad de Buenos Aires, Argentina
[4] Universidad Nacional de Quilmes, Argentina

Abstract. Call-by-Push-Value (CBPV) is a programming paradigm subsuming both Call-by-Name (CBN) and Call-by-Value (CBV) semantics. The paradigm was recently modelled by means of the Bang Calculus, a term language connecting CBPV and Linear Logic.
This paper presents a revisited version of the Bang Calculus, called $\lambda!$, enjoying some important properties missing in the original system. Indeed, the new calculus integrates commutative conversions to unblock value redexes while being confluent at the same time. A second contribution is related to non-idempotent types. We provide a quantitative type system for our $\lambda!$-calculus, and we show that the length of the (weak) reduction of a typed term to its normal form *plus* the size of this normal form is bounded by the size of its type derivation. We also explore the properties of this type system with respect to CBN/CBV translations. We keep the original CBN translation from λ-calculus to the Bang Calculus, which preserves normal forms and is sound and complete with respect to the (quantitative) type system for CBN. However, in the case of CBV, we reformulate both the translation and the type system to restore two main properties: preservation of normal forms and completeness. Last but not least, the quantitative system is refined to a *tight* one, which transforms the previous upper bound on the length of reduction to normal form plus its size into two independent *exact* measures for them.

1 Introduction

Call-by-Push-Value. The Call-by-Push-Value (CBPV) paradigm, introduced by P.B. Levy [37,38], distinguishes between values and computations under the slogan *"a value is, a computation does"*. It subsumes the λ-calculus by adding some primitives that allow to capture both the Call-by-Name (CBN) and Call-by-Value (CBV) semantics. CBN is a lazy strategy that consumes arguments without any preliminary evaluation, potentially duplicating work, while CBV is greedy, always computing arguments disregarding whether they are used or not, which may prevent a normalising term from terminating, *e.g.* $(\lambda x.I)\,\Omega$, where $I = \lambda x.x$ and $\Omega = (\lambda x.x\,x)\,(\lambda x.x\,x)$.

Essentially, CBPV introduces unary primitives `thunk` and `force`. The former freezes the execution of a term (*i.e.* it is not allowed to compute under a `thunk`) while the latter fires again a frozen term. Informally, $\texttt{force}\,(\texttt{thunk}\,t)$

is semantically equivalent to t. Resorting to the paradigm slogan, `thunk` turns a computation into a value, while `force` does the opposite. Thus, CBN and CBV are captured by conveniently labelling a λ-term using `force` and `thunk` to pause/resume the evaluation of a subterm depending on whether it is an argument (CBN) or a function (CBV). In doing so, CBPV provides a unique formalism capturing two distinct λ-calculi strategies, thus allowing to study operational and denotational semantics of CBN and CBV in a unified framework.

Bang calculus. T. Ehrhard [25] introduces a typed calculus, that can be seen as a variation of CBPV, to establish a relation between this paradigm and Linear Logic (LL). A simplified version of this formalism is later dubbed Bang calculus [26], showing in particular how CBPV captures the CBN and CBV semantics of λ-calculus via Girard's translations of intuitionistic logic into LL. A further step in this direction [15] uses Taylor expansion [27] in the Bang Calculus to approximate terms in CBPV. The Bang calculus is essentially an extension of λ-calculus with two new constructors, namely *bang* (!) and *dereliction* (der), together with the reduction rule $\mathrm{der}\,(!\,t) \mapsto t$. There are two notions of reduction for the Bang calculus, depending on whether it is allowed to reduce under a bang constructor or not. They are called *strong* and *weak reduction* respectively. Indeed, it is weak reduction that makes bang/dereliction play the role of the primitives `thunk`/`force`. Hence, these modalities are essential to capture the essence behind the CBN–CBV duality. A similar approach appears in [42], studying (simply typed) CBN and CBV translations into a fragment of IS4, recast as a very simple λ-calculus equipped with an indeterminate lax monoidal comonad.

Non-Idempotent Types. Intersection types, pioneered by [16,17], can be seen as a syntactical tool to denote programs. They are invariant under the equality generated by the evaluation rules, and type all and only all normalising terms. They are originally defined as *idempotent* types, so that the equation $\sigma \cap \sigma = \sigma$ holds, thus preventing any use of the intersection constructor to count resources. On the other hand, *non-idempotent* types, pioneered by [28], are inspired from LL, they can be seen as a syntactical formulation of its relational model [30,11]. This connection suggests a *quantitative* typing tool, being able to specify properties related to the consumption of resources, a remarkable investigation pioneered by the seminal de Carvalho's PhD thesis [18] (see also [20]). Non-idempotent types have also been used to provide characterisations of complexity classes [8]. Several papers explore the qualitative and quantitative aspects of non-idempotent types for different higher order languages, as for example Call-by-Name, Call-by-Need and Call-by-Value λ-calculi, as well as extensions to Classical Logic. Some references are [13,24,4,3,36]. Other relational models were directly defined in the more general context of LL, rather than in the λ-calculus [19,33,22,21].

An interesting recent research topic concerns the use of non-idempotent types to provide *bounds* of reduction lengths. More precisely, the size of type derivations has often been used as an *upper bound* to the length of different evaluation strategies [40,24,34,13,35,36]. A key notion behind these works is that when t

evaluates to t', then the size of the type derivation of t' is smaller than the one of t, thus the size of type derivations provides an upper bound for the *length* of the reduction to a normal form as well as for the *size* of this normal form.

A crucial point to obtain *exact bounds*, instead of upper bounds, is to consider only *minimal* type derivations, as the ones in [18,9,22]. Another approach was taken in [1], which uses an appropriate notion of *tightness* to implement minimality, a technical tool adapted to Call-by-Value [31,3] and Call-by-Need [4].

1.1 Contributions and Related Works

This paper presents a reformulation of the untyped Bang calculus, and proposes a quantitative study of it by means of non-idempotent types.

The Untyped Reduction. The Bang calculus in [25] suffers from the absence of *commutative conversions* [41,14], making some redexes to be syntactically blocked when open terms are considered. A consequence of this approach is that there are some normal forms that are semantically equivalent to non-terminating programs, a situation which is clearly unsound. This is repaired in [26] by adding commutative conversions specified by means of σ-reduction rules, which are crucial to unveil hidden (value) redexes. However, this approach presents a major drawback since the resulting combined reduction relation is not confluent.

Our revisited Bang calculus, called $\lambda!$, fixes these two problems at the same time. Indeed, the syntax is enriched with explicit substitutions, 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. This approach restores *confluence.*

The Untyped CBN and CBV Encodings. CBN and CBV (untyped) translations are extensively studied in [32,23,39]. The authors establish two encodings *cbn* and *cbv*, from untyped λ-terms into untyped terms of the Bang calculus, such that when t reduces to u in CBN (resp. CBV), $cbn(t)$ reduces to $cbn(u)$ (resp. $cbv(t)$ reduces to $cbv(u)$) in the Bang calculus. However, CBV normal forms in λ-calculus are not necessarily translated to normal forms in the Bang calculus.

Our revisited notion of reduction naturally encodes (weak) CBN as well as (open) CBV. More precisely, the $\lambda!$-calculus encodes weak CBN and open CBV specified by means of explicit substitutions, which in turn encode the corresponding well-known notions of weak CBN and open CBV (see for example [6]). These two notions are dual: weak CBN forbids reduction inside arguments, which are translated to bang terms, while open CBV forbids reduction under λ-abstractions, also translated to bang terms. More precisely, we simply extend to explicit substitutions the original CBN translation from λ-calculus to the Bang calculus, which preserves normal forms, but we subtly reformulate the CBV one. In contrast to [32], our CBV translation does preserve normal forms.

The Typed System. Starting from the relational model for the Bang calculus proposed in [32], we propose a type system for the $\lambda!$-calculus, called $\mathcal{U}$, based on non-idempotent intersection types. System $\mathcal{U}$ is able to fully *characterise* normalisation, in the sense that a term t is $\mathcal{U}$-typable if and only if t is normalising. More interestingly, we show that system $\mathcal{U}$ has also a quantitative flavour, 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. We show that system $\mathcal{U}$ also captures the standard non-idempotent intersection type system for CBN, as well as a new type system $\mathcal{V}$ that we define in this paper for CBV. System $\mathcal{V}$ characterises termination of open CBV, in the sense that t is typable in $\mathcal{V}$ if and only if t is terminating in open CBV. This can be seen as another (collateral) contribution of this paper. Moreover, the CBV embedding in [32] is not complete with respect to their type system for CBV. System $\mathcal{V}$ recovers completeness (left as an open question in [32]). Finally, an alternative CBV encoding of typed terms is proposed. This encoding is not only sound and complete, but now enjoys preservation of normal-forms.

A Refinement of the Type System Based on Tightness. A major observation concerning β-reduction in λ-calculus (and therefore in the Bang calculus) is that the size of normal forms can be exponentially bigger than the number of steps to these normal forms. This means that bounding the sum of these two integers *at the same* time is too rough, not very relevant from a quantitative point of view. Following ideas in [18,9,1], we go beyond upper bounds. Indeed, another major contribution of this paper is the refinement of the non-idempotent type system $\mathcal{U}$ to another type system $\mathcal{E}$, equipped with constants and counters, together with an appropriate notion of *tightness* (*i.e.* minimality). This new formulation fully exploits the quantitative aspect of the system, in such a way that *upper bounds* provided by system $\mathcal{U}$ are refined now into *independent exact bounds* for time and space. More precisely, given a tight type derivation Φ with counters (b, e, s) for a term t, we can show that t is normalisable in $(b + e)$-steps and its normal form has *size* s. The opposite direction also holds. Therefore, exact measures concerning the *dynamic* behaviour of t, are extracted from a *static* (tight) typing property of t.

Road-map. Sec. 2 introduces the $\lambda!$-calculus. Sec. 3 presents the sound and complete type system $\mathcal{U}$. Sec. 4 discusses (untyped and typed) CBN/CBV translations. In Sec. 5 we refine system $\mathcal{U}$ into system $\mathcal{E}$, and we prove soundness and completeness. Conclusions and future work are discussed in Sec. 6. Most of the proofs are omitted and can be found in [12].

2 The Bang Calculus Revisited

This section presents a revisited (conservative) extension of the original Bang calculi [25,26], called $\lambda!$. From a syntactical point of view, we just add explicit substitutions operators. From an operational point of view, we use *reduction at a distance* [5], thus integrating commutative conversions without jeopardising confluence (see the discussion below).

Given a countably infinite set $\mathcal{X}$ of variables $x, y, z, \ldots$ we consider the following grammar for terms (denoted by $\mathcal{T}$) and contexts:

$$
\begin{array}{rl}
\textbf{(Terms)} & t, u ::= x \in \mathcal{X} \mid t\,u \mid \lambda x.t \mid !\,t \mid \operatorname{der} t \mid t[x\backslash u] \\
\textbf{(List contexts)} & \mathtt{L} ::= \Box \mid \mathtt{L}[x\backslash t] \\
\textbf{(Contexts)} & \mathtt{C} ::= \Box \mid \mathtt{C}\,t \mid t\,\mathtt{C} \mid \lambda x.\mathtt{C} \mid !\,\mathtt{C} \mid \operatorname{der}\mathtt{C} \mid \mathtt{C}[x\backslash u] \mid t[x\backslash \mathtt{C}] \\
\textbf{(Weak contexts)} & \mathtt{W} ::= \Box \mid \mathtt{W}\,t \mid t\,\mathtt{W} \mid \lambda x.\mathtt{W} \mid \operatorname{der}\mathtt{W} \mid \mathtt{W}[x\backslash u] \mid t[x\backslash \mathtt{W}]
\end{array}
$$

Terms of the form $t[x\backslash u]$ are ***closures***, and $[x\backslash u]$ is called 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 $\Omega = \Delta\,!\,\Delta$. Weak contexts do not allow the symbol $\Box$ to occur inside the bang construct. This is similar to weak contexts in λ-calculus, where $\Box$ cannot occur inside λ-abstractions. We will see in Sec. 4 that weak reduction in the $\lambda!$-calculus perfectly captures head reduction in CBN, disallowing reduction inside arguments, as well as open CBV, disallowing reduction inside abstractions. We use $\mathtt{C}\langle t\rangle$ (resp. $\mathtt{W}\langle t\rangle$ and $\mathtt{L}\langle t\rangle$) for the term obtained by replacing the hole $\Box$ of $\mathtt{C}$ (resp. $\mathtt{W}$ and $\mathtt{L}$) by t. The notions of ***free*** and ***bound*** variables are defined as expected, in particular, $\mathtt{fv}(t[x\backslash u]) \overset{def}{=} \mathtt{fv}(t) \setminus \{x\} \cup \mathtt{fv}(u)$, $\mathtt{fv}(\lambda x.t) \overset{def}{=} \mathtt{fv}(t) \setminus \{x\}$, $\mathtt{bv}(t[x\backslash u]) \overset{def}{=} \mathtt{bv}(t) \cup \{x\} \cup \mathtt{bv}(u)$ and $\mathtt{bv}(\lambda x.t) \overset{def}{=} \mathtt{bv}(t) \cup \{x\}$. We extend the standard notion of ***α-conversion*** [7] to ES, as expected. We use $t\,\{x\backslash u\}$ to denote the ***meta-level*** substitution operation, *i.e.* all the free occurrences of the variable x in the term t are replaced by u. This operation is defined, as usual, modulo α-conversion. We use two special predicates to distinguish abstractions and bang terms possibly affected by a list of explicit substitutions. Indeed, $\mathsf{abs}(t)$ iff $t = \mathtt{L}\langle \lambda x.t'\rangle$ and $\mathsf{bang}(t)$ iff $t = \mathtt{L}\langle !\,t'\rangle$. Finally, we define the ***w-size*** of terms as follows: $|x|_w := 0$, $|t\,u|_w := 1 + |t|_w + |u|_w$, $|\lambda x.t|_w := 1 + |t|_w$, $|!\,t|_w := 0$, $|\operatorname{der} t|_w := 1 + |t|_w$, and $|t[x\backslash u]|_w := |t|_w + |u|_w$.

The ***$\lambda!$-calculus*** is given by the set of terms $\mathcal{T}$ and the ***(weak) reduction relation*** $\rightarrow_{\mathtt{w}}$, which is defined as the *union* of $\rightarrow_{\mathtt{dB}}$ ($\mathtt{d}$istant $\mathtt{B}$eta), $\rightarrow_{\mathtt{s!}}$ ($\mathtt{s}$ubstitute bang) and $\rightarrow_{\mathtt{d!}}$ ($\mathtt{d}$istant bang), defined respectively as the closure by contexts $\mathtt{W}$ of the following rewriting rules:

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

We assume that all these rules avoid capture of free variables.

Example 1. Let $t_0 = \operatorname{der}(!\,\mathtt{K})\,(!\,\mathtt{I})\,(!\,\Omega)$. Then,

$$
t_0 \rightarrow_{\mathtt{d!}} \mathtt{K}\,(!\,\mathtt{I})\,(!\,\Omega) \rightarrow_{\mathtt{dB}} (\lambda y.x)[x\backslash !\,\mathtt{I}]\,(!\,\Omega) \rightarrow_{\mathtt{dB}} x[y\backslash !\,\Omega][x\backslash !\,\mathtt{I}] \rightarrow_{\mathtt{s!}} x[x\backslash !\,\mathtt{I}] \rightarrow_{\mathtt{s!}} \mathtt{I}
$$

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

Given the translation of the Bang Calculus into LL proof-nets [25], we refer to $\mathtt{dB}$-steps as $\mathtt{m}$-steps ($\mathtt{m}$ultiplicative) and $(\mathtt{s!}, \mathtt{d!})$-steps as $\mathtt{e}$-steps ($\mathtt{e}$xponential).

Remark that reduction is *at a distance*, in the sense that the list context $\mathtt{L}$ in the rewriting rules allows the main constructors involved in these 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, in contrast to [26] which treats these conversions by means of a set of independent σ-rewriting rules, thus inheriting many drawbacks. More precisely, in the first formulation of the Bang calculus [25], there are hidden (value) redexes that block reduction, thus creating a mismatch between normal terms that are semantically non-terminating. The second formulation in [26] recovers soundness, by integrating a notion of σ-equivalence which is crucial to unveil hidden redexes and ill-formed terms (called *clashes*)[5]. However, adding σ-reduction to the logical reduction rules does not preserve confluence. Our notion of reduction addresses this two issues at the same time[6]: it integrates commutative conversions and is confluent (Thm. 1).

We write $\twoheadrightarrow_{\mathtt{w}}$ for the reflexive-transitive closure of $\rightarrow_{\mathtt{w}}$. We write $t \twoheadrightarrow_{\mathtt{w}}^{(b,e)} u$ if $t \twoheadrightarrow_{\mathtt{w}} u$ using b $\mathtt{dB}$ steps and e $(\mathtt{s!}, \mathtt{d!})$-steps.

The reduction relation $\rightarrow_{\mathtt{w}}$ enjoys a weak diamond property, *i.e.* one-step divergence can be closed in one step if the diverging terms are different, since $\rightarrow_{\mathtt{w}}$ is not reflexive. Otherwise stated, the reflexive closure of $\rightarrow_{\mathtt{w}}$ enjoys the strong diamond property.

Lemma 1. *If $t \rightarrow_{p_1} t_1$ and $t \rightarrow_{p_2} t_2$ where $t_1 \neq t_2$ and $p_1, p_2 \in \{\mathtt{dB}, \mathtt{s!}, \mathtt{d!}\}$, then there exists t_3 such that $t_1 \rightarrow_{p_2} t_3$ and $t_2 \rightarrow_{p_1} t_3$.*

The result above does not hold if reductions are allowed inside arbitrary contexts. Consider for instance the term $t = (x\,!\,x)[x\backslash !\,(\mathtt{I}\,!\,\mathtt{I})]$. We have $t \rightarrow_{\mathtt{s!}} (\mathtt{I}\,!\,\mathtt{I})\,!\,(\mathtt{I}\,!\,\mathtt{I})$ and, if we allow the reduction of the $\mathtt{dB}$-redex $\mathtt{I}\,!\,\mathtt{I}$ appearing banged inside the explicit substitution, we get $t \rightarrow_{\mathtt{dB}} (x\,!\,x)[x\backslash !\,z[z\backslash !\,\mathtt{I}]]$. Now, the term $(x\,!\,x)[x\backslash !\,z[z\backslash !\,\mathtt{I}]]$ $\mathtt{s!}$-reduces to $z[z\backslash !\,\mathtt{I}]\,!\,(z[z\backslash !\,\mathtt{I}])$, whereas two $\mathtt{dB}$-reductions are needed in order to close the diamond, *i.e.* to rewrite $(\mathtt{I}\,!\,\mathtt{I})\,!\,(\mathtt{I}\,!\,\mathtt{I})$ into $z[z\backslash !\,\mathtt{I}]\,!\,(z[z\backslash !\,\mathtt{I}])$.

This gives the following two major results (see [12] for details).

Theorem 1.

- *The reduction relation $\rightarrow_{\mathtt{w}}$ is confluent.*
- *Any two different reduction paths to normal form have the same length.*

As explained above, the strong property expressed in the second item of Thm. 1 relies essentially on the fact that reductions are disallowed under bangs.

Normal forms and neutral terms. A term is said to be $\mathtt{w}$-***normal*** if there is no t' such that $t \rightarrow_{\mathtt{w}} t'$, in which case we write $t \not\rightarrow_{\mathtt{w}}$. This notion can be characterised by means of the following inductive grammars:

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

[5] Indeed, there exist clash-free terms in normal form that are σ-reducible to normal terms with clashes, *e.g.* $R = \mathrm{der}\,((\lambda y.\lambda x.z)\,(\mathrm{der}\,(y)\,y)) \equiv_\sigma \mathrm{der}\,(\lambda x.(\lambda y.z)\,(\mathrm{der}\,(y)\,y))$.

[6] In particular, the term R is not in normal form in our framework, and it reduces to a clash term in normal form which is filtered by the type system

All these terms are $\mathtt{w}$-normal. Moreover, ***neutral*** terms do not produce any kind of redexes when inserted into a context, while ***neutral-abs*** terms (resp. ***neutral-bang***) may only produce $\mathtt{s!}$ or $\mathtt{d!}$ redexes (resp. $\mathtt{dB}$ redexes) when inserted into a context.

The intended meaning of the sets of terms defined above may be summarised as follows:

$$\mathsf{ne}_{\mathtt{w}} = \mathsf{na}_{\mathtt{w}} \cap \mathsf{nb}_{\mathtt{w}} \subset \mathsf{na}_{\mathtt{w}} \cup \mathsf{nb}_{\mathtt{w}} = \mathsf{no}_{\mathtt{w}}$$

Remark 1. Consider $t \in \mathcal{T}$. Then $t \in \mathsf{na}_{\mathtt{w}}$ and $t \in \mathsf{nb}_{\mathtt{w}}$ iff $t \in \mathsf{ne}_{\mathtt{w}}$, $t \in \mathsf{na}_{\mathtt{w}}$ and $t \notin \mathsf{nb}_{\mathtt{w}}$ implies $\mathsf{bang}(t)$, and $t \in \mathsf{nb}_{\mathtt{w}}$ and $t \notin \mathsf{na}_{\mathtt{w}}$ implies $\mathsf{abs}(t)$.

Proposition 1 (Normal Forms). *Let $t \in \mathcal{T}$. Then $t \not\to_{\mathtt{w}}$ iff $t \in \mathsf{no}_{\mathtt{w}}$.*

Clashes. Some ill-formed terms are not redexes but they don't represent a desired result for a computation either. They are called ***clashes*** (meta-variable c), and defined as follows:

$$\mathtt{L}\langle !\, t\rangle\, u \qquad t[y\backslash \mathtt{L}\langle \lambda x.u\rangle] \qquad \mathrm{der}\,(\mathtt{L}\langle \lambda x.u\rangle) \qquad t\,(\mathtt{L}\langle \lambda x.u\rangle)$$

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. In the fourth kind of clash, however, this is not the case since $t\,(\mathtt{L}\langle !\, u\rangle)$ is not a redex in general.

A term is ***clash free*** if it does not reduce to a term containing a clash, it is ***weak clash free***, written $\mathtt{wcf}$, if it does not reduce to a term containing a clash outside the scope of any constructor $!$. In other words, t is not $\mathtt{wcf}$ if and only if there exist a weak context $\mathtt{W}$ and a clash c such that $t \twoheadrightarrow_{\mathtt{w}} \mathtt{W}\langle c\rangle$.

Weak clash-free normal terms can be characterised as follows:

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

Intuitively, $\mathsf{no}_{\mathtt{wcf}}$ denotes $\mathsf{no}_{\mathtt{w}} \cap \mathtt{wcf}$ (respectively for $\mathsf{ne}_{\mathtt{wcf}}$, $\mathsf{na}_{\mathtt{wcf}}$ and $\mathsf{nb}_{\mathtt{wcf}}$).

Proposition 2 (Clash-free). *Let $t \in \mathcal{T}$. Then t is a weak clash-free normal form iff $t \in \mathsf{no}_{\mathtt{wcf}}$.*

3 The Type System $\mathcal{U}$

This section introduces a first type system $\mathcal{U}$ for our revisited version of the Bang calculus, which extends the one in [32] to explicit substitutions. We show in this paper that $\mathcal{U}$ does not only qualitatively characterise normalisation, but is also *quantitative*, in the sense that the length of the (weak) reduction of a typed term to its normal form plus the size of this normal form is bounded by the size of its type derivation. We also explore in Sec. 4 the properties of this type system with respect to the CBN and CBV translations.

Given a countable infinite set $\mathcal{TV}$ of base type variables $\alpha, \beta, \gamma, \ldots$, we define the following sets of types:

$$\textbf{(Types)}\ \sigma, \tau ::= \alpha \in \mathcal{TV} \mid \mathcal{M} \mid \mathcal{M} \to \sigma$$
$$\textbf{(Multiset types)}\ \mathcal{M} ::= [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set}$$

The empty multiset is denoted by $[\,]$. Also, $|\mathcal{M}|$ denotes the size of the multiset, thus if $\mathcal{M} = [\sigma_i]_{i \in I}$ then $|\mathcal{M}| = \#(I)$.

Typing contexts (or just ***contexts***), written Γ, Δ, are functions from variables to multiset types, assigning the empty multiset to all but a finite set of variables. The domain of Γ is given by $\mathtt{dom}(\Gamma) \overset{def}{=} \{x \mid \Gamma(x) \neq [\,]\}$. The ***union of contexts***, written $\Gamma + \Delta$, is defined by $(\Gamma + \Delta)(x) \overset{def}{=} \Gamma(x) \sqcup \Delta(x)$, where $\sqcup$ denotes multiset union. 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 (when $I = \emptyset$ the notation is to be understood as the empty context). 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 $\Gamma \vdash t : \sigma$, where Γ is a typing context, t is a term and σ is a type. The type system $\mathcal{U}$ for the $\lambda!$-calculus is given in Fig. 1.

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash t : \sigma \quad \Delta \vdash u : \Gamma(x)}{(\Gamma \setminus\!\!\setminus x) + \Delta \vdash t[x \backslash u] : \sigma}\ (\mathtt{es}) \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \setminus\!\!\setminus x \vdash \lambda x.t : \Gamma(x) \to \tau}\ (\mathtt{abs})$$

$$\frac{\Gamma \vdash t : \mathcal{M} \to \tau \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t\,u : \tau}\ (\mathtt{app}) \qquad \frac{(\Gamma_i \vdash t : \sigma_i)_{i \in I}}{+_{i \in I}\ \Gamma_i \vdash !\,t : [\sigma_i]_{i \in I}}\ (\mathtt{bg}) \qquad \frac{\Gamma \vdash t : [\sigma]}{\Gamma \vdash \operatorname{der} t : \sigma}\ (\mathtt{dr})$$

Fig. 1. System $\mathcal{U}$ for the $\lambda!$-calculus.

The axiom ($\mathtt{ax}$) is relevant (there is no weakening) and the rules ($\mathtt{app}$) and ($\mathtt{es}$) are multiplicative. Note that the argument of a bang is typed $\#(I)$ times by the premises of rule ($\mathtt{bg}$). A particular case is when $I = \emptyset$: the subterm t occurring in the typed term $!\,t$ turns out to be untyped.

A ***(type) derivation*** is a tree obtained by applying the (inductive) typing rules of system $\mathcal{U}$. The notation $\rhd_{\mathcal{U}}\ \Gamma \vdash t : \sigma$ means there is a derivation of the judgement $\Gamma \vdash t : \sigma$ in system $\mathcal{U}$. The term t is typable in system $\mathcal{U}$, or $\mathcal{U}$-typable, iff there are Γ and σ such that $\rhd_{\mathcal{U}}\ \Gamma \vdash t : \sigma$. We use the capital Greek letters $\Phi, \Psi, \ldots$ to name type derivations, by writing for example $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \sigma$. The ***size of the derivation*** Φ, denoted by $\mathtt{sz}(\Phi)$, is defined as the number of rules in the type derivation Φ except rule ($\mathtt{es}$). Note in particular that given a derivation Φ_t for a term t we always have $\mathtt{sz}(\Phi_t) \geq |t|_w$.

Example 2. The following tree Φ_0 is a type derivation for term t_0 of Example 1.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{$(\mathtt{ax})$}
\UnaryInfC{$x : [[\tau] \to \tau] \vdash x : [\tau] \to \tau$}
\RightLabel{$(\mathtt{abs})$}
\UnaryInfC{$x : [[\tau] \to \tau] \vdash \lambda y.x : [\,] \to [\tau] \to \tau$}
\RightLabel{$(\mathtt{abs})$}
\UnaryInfC{$\vdash \lambda x.\lambda y.x : [[\tau] \to \tau] \to [\,] \to [\tau] \to \tau$}
\RightLabel{$(\mathtt{bg})$}
\UnaryInfC{$\vdash \,!\,\mathtt{K} : [[[\tau] \to \tau] \to [\,] \to [\tau] \to \tau]$}
\RightLabel{$(\mathtt{dr})$}
\UnaryInfC{$\vdash \mathrm{der}\,(!\,\mathtt{K}) : [[\tau] \to \tau] \to [\,] \to [\tau] \to \tau$}
\AxiomC{}
\RightLabel{$(\mathtt{ax})$}
\UnaryInfC{$x : [\tau] \vdash x : \tau$}
\RightLabel{$(\mathtt{abs})$}
\UnaryInfC{$\vdash \lambda x.x : [\tau] \to \tau$}
\RightLabel{$(\mathtt{bg})$}
\UnaryInfC{$\vdash \,!\,\mathtt{I} : [[\tau] \to \tau]$}
\RightLabel{$(\mathtt{app})$}
\BinaryInfC{$\vdash \mathrm{der}\,(!\,\mathtt{K})\,(!\,\mathtt{I}) : [\,] \to [\tau] \to \tau$}
\AxiomC{}
\RightLabel{$(\mathtt{bg})$}
\UnaryInfC{$\vdash \,!\,\Omega : [\,]$}
\RightLabel{$(\mathtt{app})$}
\BinaryInfC{$\vdash \mathrm{der}\,(!\,\mathtt{K})\,(!\,\mathtt{I})\,(!\,\Omega) : [\tau] \to \tau$}
\end{prooftree}
$$

Note that $\mathtt{sz}(\Phi_0) = 11$.

Typable terms are necessarily weak clash-free:

Lemma 2. *If* $\rhd_{\mathcal{U}} \Gamma \vdash t : \sigma$*, then* t *is* $\mathtt{wcf}$*.*

Proof. Assume towards a contradiction that t is not $\mathtt{wcf}$, *i.e.* there exists a weak context $\mathtt{W}$ and a clash c such that $t \twoheadrightarrow_{\mathtt{w}} \mathtt{W}\langle c \rangle$. Then, Lem. 3 (WSR) gives $\Phi' \rhd_{\mathcal{U}} \Gamma \vdash \mathtt{W}\langle c \rangle : \sigma$. If we show that a term of the form $\mathtt{W}\langle c \rangle$ cannot be typed in system $\mathcal{U}$, we are done. This follows by straightforward induction on $\mathtt{W}$. The base case is when $\mathtt{W} = \Box$. For every possible c, it is immediate to see that there is a mismatch between its syntactical form and the typing rules of system $\mathcal{U}$. For instance, if $c = \mathtt{L}\langle !\,t \rangle\, u$, then $\mathtt{L}\langle !\,t \rangle$ should have a functional type by rule $(\mathtt{app})$ but it can only be assigned a multiset type by rules $(\mathtt{es})$ and $(\mathtt{bg})$. As for the inductive case, an easy inspection of the typing rules shows for all terms t and weak contexts $\mathtt{W}$, t must be typed in order to type $\mathtt{W}\langle t \rangle$. $\Box$

However, normal terms are not necessarily clash-free, but the type system captures clash-freeness of normal terms. Said differently, when restricted to $\mathsf{no}_{\mathtt{w}}$, typability exactly corresponds to weak clash-freeness.

Theorem 2. *Let* $t \in \mathcal{T}$*. Then,* $t \in \mathsf{no}_{\mathtt{wcf}}$ *iff* $t \in \mathsf{no}_{\mathtt{w}}$ *and* t *is* $\mathcal{U}$*-typable.*

The quantitative aspect of system $\mathcal{U}$ is materialised in the following weighted subject reduction (WSR) and expansion (WSE) properties.

Lemma 3. *Let* $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \tau$*.*

(WSR) *If* $t \to_{\mathtt{w}} t'$*, then there is* $\Phi' \rhd_{\mathcal{U}} \Gamma \vdash t' : \tau$ *such that* $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$*.*
(WSE) *If* $t' \to_{\mathtt{w}} t$*, then there is* $\Phi' \rhd_{\mathcal{U}} \Gamma \vdash t' : \tau$ *such that* $\mathtt{sz}(\Phi') > \mathtt{sz}(\Phi)$*.*

Erasing steps like $y[x \backslash !\, z] \to_{\mathtt{s!}} y$ may seem problematic for subject reduction and expansion, but they are not: the variable x, as well as $!\,z$ are necessarily both typed with $[\,]$, so there is no loss of information since the contexts allowing to type the redex and the reduced term are the same.

Typability can then be shown to (qualitatively and quantitatively) characterise normalisation.

Theorem 3 (Soundness and Completeness). *The term t is $\mathcal{U}$-typable iff t $\mathtt{w}$-normalises to a term $p \in \mathsf{no}_{\mathtt{wcf}}$. Moreover, if $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \tau$, then $t \twoheadrightarrow_{\mathtt{w}}^{(b,e)} p$ and $\mathtt{sz}(\Phi) \geq b + e + |p|_w$.*

Proof. The soundness proof is straightforward by Lem. 3 (WSR) and Thm. 2. Remark that the argument is simply combinatorial, no reducibility argument is needed. Moreover, unlike [26,32], there is no need to reason through any intermediate *resource* Bang calculus. For the completeness proof, we reason by induction on the length of the $\mathtt{w}$-normalising sequence. For the base case, we use Thm. 2 which states that $p \in \mathsf{no}_{\mathtt{wcf}}$ implies p is $\mathcal{U}$-typable. For the inductive case we use Lem. 3 (WSE). The *moreover* statement holds by Lem. 3 and the fact that the size of the type derivation of p is greater than or equal to $|p|_w$. $\square$

The previous theorem can be illustrated by the term $t_0 = \mathrm{der}\,(!\,\mathtt{K})\,(!\,\mathtt{I})\,(!\,\Omega)$ defined in Example 1, which normalises in 5 steps to a normal form of w-size 1, the sum of the two being bounded by the size 8 of its type derivation Φ_0 given in Example 2.

4 Capturing Call-by-Name and Call-by-Value

This section explores the CBN/CBV embeddings into the $\lambda!$-calculus. For CBN, we slightly adapt Girard's translation into LL [29], which preserves normal forms and is sound and complete with respect to the standard (quantitative) type system [28]. For CBV, however, we reformulate both the translation and the type system, so that preservation of normal forms and completeness are restored. In both cases, we specify the operational semantics of CBN and CBV by means of a very simple notion of explicit substitution, see for example [6].

Terms ($\mathcal{T}_\lambda$), values and contexts are defined as follows:

$$\begin{array}{rl}
\textbf{(Terms)}\ t,u & ::= v \mid t\,u \mid t[x\backslash u] \\
\textbf{(Values)}\ v & ::= x \in \mathcal{X} \mid \lambda x.t \\
\textbf{(List Contexts)}\ \mathtt{L} & ::= \Box \mid \mathtt{L}[x\backslash t] \\
\textbf{(Call-by-Name Contexts)}\ \mathtt{N} & ::= \Box \mid \mathtt{N}\,t \mid \lambda x.\mathtt{N} \mid \mathtt{N}[x\backslash u] \\
\textbf{(Call-by-Value Contexts)}\ \mathtt{V} & ::= \Box \mid \mathtt{V}\,t \mid t\,\mathtt{V} \mid \mathtt{V}[x\backslash u] \mid t[x\backslash \mathtt{V}]
\end{array}$$

As in Sec. 2 we use the predicate $\mathsf{abs}(t)$ iff $t = \mathtt{L}\langle \lambda x.t' \rangle$. We also use the predicates $\mathsf{app}(t)$ iff $t = \mathtt{L}\langle t'\,t'' \rangle$ and $\mathsf{var}(t)$ iff $t = \mathtt{L}\langle x \rangle$.

The ***Call-by-Name*** reduction relation $\rightarrow_{\mathtt{n}}$ is defined as the closure of contexts $\mathtt{N}$ of the rules $\mathtt{dB}$ and $\mathtt{s}$ presented below, while the ***Call-by-Value*** reduction relation $\rightarrow_{\mathtt{v}}$ is defined as the closure of contexts $\mathtt{V}$ of the rules $\mathtt{dB}$ and $\mathtt{sv}$ below. Equivalently, $\rightarrow_{\mathtt{n}} := \mathtt{N}(\mapsto_{\mathtt{dB}} \cup \mapsto_{\mathtt{s}})$ and $\rightarrow_{\mathtt{v}} := \mathtt{V}(\mapsto_{\mathtt{dB}} \cup \mapsto_{\mathtt{sv}})$ and

$$\begin{array}{rcl}
\mathtt{L}\langle \lambda x.t \rangle\, u & \mapsto_{\mathtt{dB}} & \mathtt{L}\langle t[x\backslash u] \rangle \\
t[x\backslash u] & \mapsto_{\mathtt{s}} & t\,\{x\backslash u\} \\
t[x\backslash \mathtt{L}\langle v \rangle] & \mapsto_{\mathtt{sv}} & \mathtt{L}\langle t\,\{x\backslash v\} \rangle
\end{array}$$

We write $t \not\rightarrow_{\mathtt{n}}$ (resp. $t \not\rightarrow_{\mathtt{v}}$), and call t an $\mathtt{n}$-***normal form*** (resp. $\mathtt{v}$-***normal form***), if t cannot be reduced by means of $\rightarrow_{\mathtt{n}}$ (resp. $\rightarrow_{\mathtt{v}}$).

Remark that we use CBN and CBV formulations based on distinguished multiplicative (*cf.* $\mathtt{dB}$) and exponential (*cf.* $\mathtt{s}$ and $\mathtt{sv}$) rules, inheriting the nature of cut elimination rules in LL. Moreover, CBN is to be understood as *head* CBN reduction [7], *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.

Embeddings. The CBN and CBV embeddings into the $\lambda!$-calculus, written $\_^{\mathsf{cbn}}$ and $\_^{\mathsf{cbv}}$ resp., are inductively defined as:

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

Remark that there are no two consecutive ! constructors in the image of the translations. The CBN embedding extends Girard's translation to explicit substitutions, while the CBV one is different. Indeed, the translation of an application $t\,u$ is usually defined as $\mathrm{der}\,(t^{\mathsf{cbv}})\,u^{\mathsf{cbv}}$ (see for example [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 $\mathtt{w}$-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.

Lemma 4. *Let $t \in \mathcal{T}_\lambda$. If $t \not\to_{\mathtt{n}}$, then $t^{\mathsf{cbn}} \not\to_{\mathtt{w}}$. If $t \not\to_{\mathtt{v}}$, then $t^{\mathsf{cbv}} \not\to_{\mathtt{w}}$.*

Simulation of CBN and CBV reductions in the $\lambda!$-calculus can be shown by induction on the reduction relations.

Lemma 5. *Let $t \in \mathcal{T}_\lambda$. If $t \to_{\mathtt{n}} s$, then $t^{\mathsf{cbn}} \to_{\mathtt{w}} s^{\mathsf{cbn}}$. If $t \to_{\mathtt{v}} s$, then $t^{\mathsf{cbv}} \twoheadrightarrow_{\mathtt{w}} s^{\mathsf{cbv}}$.*

Note that the CBV case may require many reduction steps between t^{cbv} and s^{cbv} and not just one. For instance, if $t = \mathtt{I}\,y\,z \to_{\mathtt{v}} w[w\backslash y]\,z = s$, then $t^{\mathsf{cbv}} = \mathrm{der}\,((\lambda w.!\,w)\,!\,y)\,!\,z \to_{\mathtt{w}} \mathrm{der}\,(!\,w[w\backslash !\,y])\,!\,z \to_{\mathtt{w}} w[w\backslash !\,y]\,!\,z = s^{\mathsf{cbv}}$.

Non-Idempotent Types for Call-by-Name and Call-by-Value. For CBN we use the non-idempotent type system defined in [35] for explicit substitutions, that we present in Fig. 2 (top), and which is an extension of that in [28]. For CBV we slightly reformulate the non-idempotent system in [32], as presented in Fig. 2 (bottom), in order to recover completeness of the (typed) CBV translation.

We write $\Phi \rhd_{\mathcal{N}} \Gamma \vdash t : \sigma$ (resp. $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : \sigma$) for a type derivation Φ in system $\mathcal{N}$ (resp. $\mathcal{V}$). Remark that potential erasable terms are typed with multiple premises: this is the case for the arguments of applications and the arguments of substitutions in CBN, as well as the values in CBV. A key point in rule $(\mathtt{app_v})$ is that left hand sides of applications are typed with multisets of the form $[\mathcal{M} \to \tau]$, where τ is any type, potentially a base one, while [32] necessarily requires a

System $\mathcal{N}$ for Call-by-Name

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathtt{ax_n}) \qquad \frac{\Gamma; x : [\sigma_i]_{i\in I} \vdash t : \tau \quad (\Delta_i \vdash u : \sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash t[x\backslash u] : \tau}\ (\mathtt{es_n})$$

$$\frac{\Gamma \vdash t : \tau}{\Gamma \setminus\!\!\setminus x \vdash \lambda x.t : \Gamma(x) \to \tau}\ (\mathtt{abs_n}) \qquad \frac{\Gamma \vdash t : [\sigma_i]_{i\in I} \to \tau \quad (\Delta_i \vdash u : \sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash t\,u : \tau}\ (\mathtt{app_n})$$

System $\mathcal{V}$ for Call-by-Value

$$\frac{}{x : \mathcal{M} \vdash x : \mathcal{M}}\ (\mathtt{ax_v}) \qquad \frac{\Gamma \vdash t : \sigma \quad \Delta \vdash u : \Gamma(x)}{(\Gamma \setminus\!\!\setminus x) + \Delta \vdash t[x\backslash u] : \sigma}\ (\mathtt{es_v})$$

$$\frac{(\Gamma_i \vdash t : \tau_i)_{i\in I}}{+_{i\in I} \Gamma_i \setminus\!\!\setminus x \vdash \lambda x.t : [\Gamma_i(x) \to \tau_i]_{i\in I}}\ (\mathtt{abs_v}) \qquad \frac{\Gamma \vdash t : [\mathcal{M} \to \tau] \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t\,u : \tau}\ (\mathtt{app_v})$$

Fig. 2. Typing schemes for CBN/CBV.

multiset of the form $[\mathcal{M} \to \mathcal{M}']$, a subtle difference which breaks completeness. System $\mathcal{N}$ (resp. $\mathcal{V}$) can be understood as a relational model of the Call-by-Name (resp. Call-by-Value) calculus, in the sense that typing is stable by reduction and expansion [12].

The CBV translation is not complete for the system in [32], *i.e.* there exist a λ-term t such that $\Gamma \vdash t^{\mathsf{cbv}} : \sigma$ is derivable in $\mathcal{U}$ but $\Gamma \vdash t : \sigma$ is not derivable in their system. This is restored in this paper. More precisely, our two embeddings are sound and complete w.r.t. system $\mathcal{U}$:

Theorem 4 (Soundness/Completeness of the Embeddings). *Let $t \in \mathcal{T}_\lambda$.*

1. $\rhd_{\mathcal{N}} \Gamma \vdash t : \sigma$ *iff* $\rhd_{\mathcal{U}} \Gamma \vdash t^{\mathsf{cbn}} : \sigma$.
2. $\rhd_{\mathcal{V}} \Gamma \vdash t : \sigma$ *iff* $\rhd_{\mathcal{U}} \Gamma \vdash t^{\mathsf{cbv}} : \sigma$.

The type system $\mathcal{N}$ (resp. $\mathcal{V}$) characterises `n`-normalisation (resp. `v`-normalisation). More precisely:

Theorem 5 (Characterisation of CBN/CBV Normalisation). *Let $t \in \mathcal{T}_\lambda$.*

- *t is $\mathcal{N}$-typable iff t is `n`-normalising.*
- *t is $\mathcal{V}$-typable iff t is `v`-normalising.*

5 A Tight Type System Giving Exact Bounds

In order to count exactly the length of `w`-reduction sequences to normal forms, we first fix a *deterministic* strategy for the λ!-calculus, called `dw`, which computes the *same* `w`-normal forms. We then define the *tight* type system $\mathcal{E}$, being able to count exactly the length of `dw`-reduction sequences. Thm. 1, stating that any two different reductions paths to normal form have the same length, guarantees that *any* `w`-reduction sequence to normal form can be exactly measured.

A Deterministic Strategy for the λ!-Calculus. The reduction relation $\rightarrow_{\mathtt{dw}}$ defined below is a deterministic version of $\rightarrow_{\mathtt{w}}$ and is used, as explained, as a technical tool of our development.

$$\frac{}{\mathtt{L}\langle\lambda x.t\rangle\, u \rightarrow_{\mathtt{dw}} \mathtt{L}\langle t[x\backslash u]\rangle} \qquad \frac{}{t[x\backslash \mathtt{L}\langle !\, u\rangle] \rightarrow_{\mathtt{dw}} \mathtt{L}\langle t\,\{x\backslash u\}\rangle} \qquad \frac{}{\mathrm{der}\,(\mathtt{L}\langle !\, t\rangle) \rightarrow_{\mathtt{dw}} \mathtt{L}\langle t\rangle}$$

$$\frac{t \rightarrow_{\mathtt{dw}} u}{\lambda x.t \rightarrow_{\mathtt{dw}} \lambda x.u} \qquad \frac{t \rightarrow_{\mathtt{dw}} u \quad \neg\mathsf{bang}(t)}{r[x\backslash t] \rightarrow_{\mathtt{dw}} r[x\backslash u]} \qquad \frac{t \rightarrow_{\mathtt{dw}} u \quad \neg\mathsf{bang}(t)}{\mathrm{der}\, t \rightarrow_{\mathtt{dw}} \mathrm{der}\, u}$$

$$\frac{t \rightarrow_{\mathtt{dw}} u \quad \neg\mathsf{abs}(t)}{t\, r \rightarrow_{\mathtt{dw}} u\, r} \qquad \frac{t \rightarrow_{\mathtt{dw}} u \quad r \in \mathsf{na}_{\mathtt{w}}}{r\, t \rightarrow_{\mathtt{dw}} r\, u} \qquad \frac{t \rightarrow_{\mathtt{dw}} u \quad r \in \mathsf{nb}_{\mathtt{w}}}{t[x\backslash r] \rightarrow_{\mathtt{dw}} u[x\backslash r]}$$

Normal forms of $\rightarrow_{\mathtt{w}}$ and $\rightarrow_{\mathtt{dw}}$ are the same, both characterised by the set $\mathsf{no}_{\mathtt{w}}$.

Proposition 3. *Let $t \in \mathcal{T}$. Then, (1) $t \not\rightarrow_{\mathtt{w}}$ iff (2) $t \not\rightarrow_{\mathtt{dw}}$ iff (3) $t \in \mathsf{no}_{\mathtt{w}}$.*

Proof. Notice that (1) $\implies$ (2) follows from $\rightarrow_{\mathtt{dw}} \subset \rightarrow_{\mathtt{w}}$. Moreover, (1) iff (3) holds by Prop. 1. The proof of (2) $\implies$ (3) follows from a straightforward adaptation of this same proposition. □

The Type System $\mathcal{E}$. We now extend the type system $\mathcal{U}$ to a *tight* one, called $\mathcal{E}$, being able to provide *exact* bounds for $\mathtt{dw}$-normalising sequences and size of normal forms. The technique is based on [1], which defines type systems to count reduction lengths for different strategies in the λ-calculus. The notion of tight derivation turns out to be a particular implementation of *minimal derivation*, pioneered by de Carvalho in [18], where exact bounds for CBN abstract machines are inferred from minimal type derivations.

We define the following sets of types:

$$\begin{array}{rl} \textbf{(Tight types)} & \mathtt{tt} ::= \mathtt{a} \mid \mathtt{b} \mid \mathtt{n} \\ \textbf{(Types)} & \sigma, \tau ::= \mathtt{tt} \mid \mathcal{M} \mid \mathcal{M} \rightarrow \sigma \\ \textbf{(Multiset types)} & \mathcal{M} ::= [\sigma_i]_{i\in I} \text{ where } I \text{ is a finite set} \end{array}$$

Inspired by [1], which only uses two constant types $\mathtt{a}$ and $\mathtt{n}$ for abstractions and neutral terms respectively, we now use three base types. Indeed, the constant $\mathtt{a}$ (resp. $\mathtt{b}$) types terms whose normal form has the shape $\mathtt{L}\langle\lambda x.t\rangle$ (resp. $\mathtt{L}\langle !\, t\rangle$), and the constant $\mathtt{n}$ types terms whose normal form is in $\mathsf{ne}_{\mathtt{wcf}}$. 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, $\overline{\mathtt{a}} \in \{\mathtt{b}, \mathtt{n}\}$.

Typing contexts are functions from variables to multiset types, assigning the empty multiset to all but a finite number of variables. Sequents are of the form $\Gamma \vdash^{(b,e,s)} t : \sigma$, where the natural numbers b, e and s provide information on the reduction of t to normal form, and on the size of its normal form. More precisely, b (resp. e) indicates the number of multiplicative (resp. exponential) steps to normal form, while s indicates the w-size of this normal form. Remark that we do not count $\mathtt{s}!$ and $\mathtt{d}!$ steps separately, because both of them are exponential steps of the same nature. It is also worth noticing that only two counters suffice in the case of the λ-calculus [1], one to count β-reduction steps, and another to count the size of normal forms. The difficulty in the case of the $\lambda!$-calculus is to statically discriminate between multiplicative and exponential steps. Typing rules (Fig. 3) are split in two groups: the *persistent* and the *consuming* ones. A constructor is consuming (resp. persistent) if it is consumed (resp. not consumed) during $\mathtt{w}$-reduction. For instance, in $\text{der}\,(!\,\mathtt{K})\,(!\,\mathtt{I})\,(!\,\Omega)$ 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. This dichotomy between consuming/persistent constructors has been used in [1] for the λ-calculus, and adapted here for the $\lambda!$-calculus.

Persistent Typing Rules

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

$$\frac{}{\vdash^{(0,0,0)} !\,t : \mathtt{b}}\ (\mathtt{bg}_\mathtt{p}) \qquad \frac{\Gamma \vdash^{(b,e,s)} t : \mathtt{n}}{\Gamma \vdash^{(b,e,s+1)} \text{der}\,t : \mathtt{n}}\ (\mathtt{dr}_\mathtt{p})$$

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

Consuming Typing Rules

$$\frac{}{x : [\sigma] \vdash^{(0,0,0)} x : \sigma}\ (\mathtt{ax}_\mathtt{c}) \qquad \frac{\Gamma \vdash^{(b,e,s)} t : \mathcal{M} \to \tau \quad \Delta \vdash^{(b',e',s')} u : \mathcal{M}}{\Gamma + \Delta \vdash^{(b+b'+1,e+e',s+s')} t\,u : \tau}\ (\mathtt{ae}_\mathtt{c1})$$

$$\frac{\Gamma \vdash^{(b,e,s)} t : \mathcal{M} \to \tau \quad \Delta \vdash^{(b',e',s')} u : \mathtt{n} \quad \mathsf{tight}(\mathcal{M})}{\Gamma + \Delta \vdash^{(b+b'+1,e+e',s+s')} t\,u : \tau}\ (\mathtt{ae}_\mathtt{c2})$$

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

$$\frac{\Gamma \vdash^{(b,e,s)} t : [\sigma]}{\Gamma \vdash^{(b,e,s)} \text{der}\,t : \sigma}\ (\mathtt{dr}_\mathtt{c}) \qquad \frac{\Gamma \vdash^{(b,e,s)} t : \sigma \quad \Delta \vdash^{(b',e',s')} u : \Gamma(x)}{(\Gamma \setminus\!\!\setminus x) + \Delta \vdash^{(b+b',e+e',s+s')} t[x\backslash u] : \sigma}\ (\mathtt{es}_\mathtt{c})$$

Fig. 3. System $\mathcal{E}$ for the $\lambda!$-Calculus.

The persistent rules 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 type consuming constructors, so that they increase the first two counters, contributing to the length of the normalisation sequence. More precisely, rules $(\mathtt{ae_{c1}})$ and $(\mathtt{ae_{c2}})$ increments the first counter because the (consuming) application will be used to perform a $\mathtt{dB}$-step, while rule $(\mathtt{bg_c})$ increments the second counter because the (consuming) bang will be used to perform either a $\mathtt{s!}$ or a $\mathtt{d!}$-step. Rule $(\mathtt{ae_{c2}})$ 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 bang.

A multiset type $[\sigma_i]_{i\in I}$ 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 multisets to all variables. A type derivation $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(b,e,s)} t : \sigma$ is ***tight*** if Γ is tight and $\sigma \in \mathtt{tt}$.

Example 3. The following tight typing can be derived for term t_0 of Example 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{a}] \vdash^{(0,0,0)} x : \mathtt{a}}\,(\mathtt{ax_c})}{x : [\mathtt{a}] \vdash^{(0,0,0)} \lambda y.x : [\,] \to \mathtt{a}}\,(\mathtt{ai_c})}{\vdash^{(0,0,0)} \lambda x.\lambda y.x : [\mathtt{a}] \to [\,] \to \mathtt{a}}\,(\mathtt{ai_c})}{\vdash^{(0,1,0)} !\,\mathtt{K} : [[\mathtt{a}] \to [\,] \to \mathtt{a}]}\,(\mathtt{bg_c})}{\vdash^{(0,1,0)} \mathrm{der}\,(!\,\mathtt{K}) : [\mathtt{a}] \to [\,] \to \mathtt{a}}\,(\mathtt{dr_c}) \quad \dfrac{\dfrac{\dfrac{}{x : [\mathtt{n}] \vdash^{(0,0,0)} x : \mathtt{n}}\,(\mathtt{ax_c})}{\vdash^{(0,0,1)} \lambda x.x : \mathtt{a}}\,(\mathtt{ai_p})}{\vdash^{(0,1,1)} !\,\mathtt{I} : [\mathtt{a}]}\,(\mathtt{bg_c})}{\vdash^{(1,2,1)} \mathrm{der}\,(!\,\mathtt{K})\,(!\,\mathtt{I}) : [\,] \to \mathtt{a}}\,(\mathtt{ae_{c1}}) \quad \dfrac{}{\vdash^{(0,1,0)} !\,\Omega : [\,]}\,(\mathtt{bg_c})}{\vdash^{(2,3,1)} \mathrm{der}\,(!\,\mathtt{K})\,(!\,\mathtt{I})\,(!\,\Omega) : \mathtt{a}}\,(\mathtt{ae_{c1}})$$

Note that the only persistent rule used is $(\mathtt{ai_p})$ when typing $\mathtt{I}$, thus contributing to count the w-size of the $\mathtt{w}$-normal form of t_0, which is $\mathtt{I}$.

Soundness. We now study soundness of the type system $\mathcal{E}$, which does not only guarantee that typable terms are normalising –a qualitative property– but also provides quantitative (exact) information for normalising sequences. More precisely, given a tight type derivation Φ with counters (b, e, s) for a term t, t is $\mathtt{w}$-normalisable in $(b + e)$-steps and its $\mathtt{w}$-normal form has size s. Therefore, information about a *dynamic* behaviour of t, is extracted from a static typing property of t. The soundness proof is mainly based on a subject reduction property (Lem. 9), as well as on some auxiliary lemmas.

As in system $\mathcal{U}$, typable terms are weak clash-free:

Lemma 6. *If $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(b,e,s)} t : \sigma$, then t is $\mathtt{wcf}$.*

The following properties can be shown by induction on the derivations.

Lemma 7. *If $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(b,e,s)} t : \sigma$ is tight, then $b = e = 0$ iff $t \in \mathsf{no}_{\mathtt{w}}$.*

Lemma 8. *If $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(0,0,s)} t : \sigma$ is tight, then $s = |t|_w$.*

The type system $\mathcal{E}$ also captures clash-freeness of normal terms:

Theorem 6. *Let $t \in \mathcal{T}$. Then $t \in \mathsf{no}_{\mathtt{w}}$ and t is $\mathcal{E}$-typable iff $t \in \mathsf{no}_{\mathtt{wcf}}$.*

To conclude soundness, the key property is subject reduction, stating that every reduction step decreases the counters of tight derivations by exactly one.

Lemma 9 (Exact Subject Reduction). *Let $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(b,e,s)} t : \sigma$ such that Γ is tight, and either $\sigma \in \mathtt{tt}$ or $\neg\mathsf{abs}(t)$. If $t \to_{\mathtt{dw}} t'$, then there is $\Phi' \rhd_{\mathcal{E}} \Gamma \vdash^{(b',e',s)} t' : \sigma$ such that*

- *$b' = b - 1$ and $e' = e$ if $t \to_{\mathtt{dw}} t'$ is an $\mathtt{m}$-step.*
- *$e' = e - 1$ and $b' = b$ if $t \to_{\mathtt{dw}} t'$ is an $\mathtt{e}$-step.*

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

Proof. We prove the statement by showing that $t \twoheadrightarrow_{\mathtt{dw}}^{(b,e)} p$ holds for the deterministic strategy, then we conclude since $\to_{\mathtt{dw}} \subseteq \to_{\mathtt{w}}$. Let $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(b,e,s)} t : \sigma$. We reason by induction on $b + e$.

If $b + e = 0$, then $b = e = 0$ and Lem. 7 gives $t \in \mathsf{no}_{\mathtt{w}}$. Moreover, by Lem. 8 and Thm. 6, we get both $|t|_w = s$ and $t \in \mathsf{no}_{\mathtt{wcf}}$. Thus, we conclude with $p = t$.

If $b + e > 0$, then $t \notin \mathsf{no}_{\mathtt{w}}$ holds by Lem. 7 and thus there exists t' such that $t \twoheadrightarrow_{\mathtt{dw}}^{(1,0)} t'$ or $t \twoheadrightarrow_{\mathtt{dw}}^{(0,1)} t'$ by Prop. 3. By Lem. 9 there is $\Phi' \rhd_{\mathcal{E}} \Gamma \vdash^{(b',e',s)} t' : \sigma$ such that $1 + b' + e' = b + e$. By the *i.h.* there is p such that $p \in \mathsf{no}_{\mathtt{wcf}}$ and $t' \twoheadrightarrow_{\mathtt{dw}}^{(b',e')} p$ with $s = |p|_w$. Then $t \twoheadrightarrow_{\mathtt{dw}}^{(1,0)} t' \twoheadrightarrow_{\mathtt{dw}}^{(b',e')} p$ (resp. $t \twoheadrightarrow_{\mathtt{dw}}^{(0,1)} t' \dots$) which means $t \twoheadrightarrow_{\mathtt{dw}}^{(b,e)} p$, as expected. □

Completeness. We now study completeness of the type system $\mathcal{E}$, which does not only guarantee that normalising terms are typable –a qualitative property– but also provides a tight type derivation having appropriate counters. More precisely, given a term t which is $\mathtt{w}$-normalisable by means of b $\mathtt{dB}$-steps and e $(\mathtt{s!}, \mathtt{d!})$-steps, and having a $\mathtt{w}$-normal form of size s, there is a tight derivation Φ for t with counters (b, e, s). The completeness proof is mainly based on a subject expansion property (Lem. 11), as well as on an auxiliary lemma providing tight derivations with appropriate counters for $\mathtt{w}$-normal weak clash-free terms.

Lemma 10. *If $t \in \mathsf{no}_{\mathtt{wcf}}$, then there is a tight derivation $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(0,0,|t|_w)} t : \sigma$.*

Lemma 11 (Exact Subject Expansion). *Let $\Phi' \rhd_{\mathcal{E}} \Gamma \vdash^{(b',e',s)} t' : \sigma$ be a tight derivation. If $t \to_{\mathtt{dw}} t'$, then there is $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(b,e,s)} t : \sigma$ such that*

- *$b' = b - 1$ and $e' = e$ if $t \to_{\mathtt{dw}} t'$ is an $\mathtt{m}$-step.*
- *$e' = e - 1$ and $b' = b$ if $t \to_{\mathtt{dw}} t'$ is an $\mathtt{e}$-step.*

Theorem 8 (Completeness). *If $t \twoheadrightarrow_{\mathtt{w}}^{(b,e)} p$ with $p \in \mathsf{no}_{\mathtt{wcf}}$, then there exists a tight type derivation $\Phi \rhd_{\mathcal{E}} \Gamma \vdash^{(b,e,|p|_w)} t : \sigma$.*

Proof. We prove the statement for $\twoheadrightarrow_{\mathtt{dw}}$ and then conclude for the general notion of reduction $\to_{\mathtt{w}}$ by Thm. 1. Let $t \twoheadrightarrow_{\mathtt{dw}}^{(b,e)} p$. We proceed by induction on $b + e$.

If $b + e = 0$, then $b = e = 0$ and thus $t = p$, which implies $t \in \mathsf{no}_{\mathtt{wcf}}$. Lem. 10 allows to conclude.

If $b + e > 0$, then there is t' such that $t \twoheadrightarrow_{\mathtt{dw}}^{(1,0)} t' \twoheadrightarrow_{\mathtt{dw}}^{(b-1,e)} p$ or $t \twoheadrightarrow_{\mathtt{dw}}^{(0,1)} t' \twoheadrightarrow_{\mathtt{dw}}^{(b,e-1)} p$. By the *i.h.* there is a tight derivation $\Phi' \triangleright_{\mathcal{E}} \Gamma \vdash^{(b',e',|p|_w)} t' : \sigma$ such $b' + e' = b + e - 1$. Lem. 11 gives a tight derivation $\Phi \triangleright_{\mathcal{E}} \Gamma \vdash^{(b'',e'',|p|_w)} t : \sigma$ such $b'' + e'' = b' + e' + 1$. We then have $b'' + e'' = b + e$. The fact that $b'' = b$ and $e'' = e$ holds by a simple case analysis. □

The main results can be illustrated by the term $t_0 = \mathrm{der}\,(!\,\mathtt{K})\,(!\,\mathtt{I})\,(!\,\Omega)$ in Sec. 2, which normalises in 2 multiplicative steps and 3 exponential steps to a normal form of w-size 1. A tight derivation for t_0 with appropriate counters $(2, 3, 1)$ is given in Example 3.

6 Conclusion

This paper gives a fresh view of the Bang Calculus, a formalism introduced by T. Ehrhard to study the relation between CBPV and Linear Logic.

Our reduction relation integrates commutative conversions inside the logical original formulation of [25], thus recovering soundness, *i.e.* avoiding mismatches between terms in normal form that are semantically non-terminating. In contrast to [26], which models commutative conversions as σ-reduction rules by paying the cost of loosing confluence, our *at a distance* formulation yields a confluent reduction system.

We then define two non-idempotent type systems for our calculus. System $\mathcal{U}$ provides upper bounds for the length of normalising sequences plus the size of normal forms. Moreover, it captures typed CBN and CBV. In particular, we reformulate the translation and the type system of CBV to restore two major properties missing in [32]: preservation of normal forms and completeness. Last but not least, the quantitative system $\mathcal{U}$ is further refined into system $\mathcal{E}$, being able to provide *exact* bounds for normalising sequences and size of normal forms, independently. Moreover, our tight system $\mathcal{E}$ is able to *discriminate* between different kind of steps performed to normalise terms.

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 $!$. Another challenging problem is to relate *tight* typing in CBN/CBV with *tight* typing in our calculus, thus providing an exact correspondence between (CBN/CBV) reduction steps and $\lambda!$-reduction steps.

Acknowledgments

We are grateful to Giulio Guerrieri and Giulio Manzonetto for fruitful discussions. This work has been partially supported by LIA INFINIS/IRP SINFIN, the ECOS-Sud program PA17C01, and the ANR COCA HOLA.

References

1. Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *PACMPL*, 2 (ICFP):94:1–94:30, 2018.
2. Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. *Asian Programming Languages and Systems Symposium* (APLAS), Hanoi, Vietnam, LNCS 10017, pages 206–226, 2016.
3. Beniamino Accattoli and Giulio Guerrieri. Types of fireballs. *Asian Programming Languages and Systems Symposium* (APLAS), Wellington, New Zealand, LNCS 11275, pages 45–66. Springer, 2018.
4. Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. *Programming Languages and Systems European Symposium on Programming* (ESOP), Prague, Czech Republic, LNCS 11423, pages 410–439. Springer, 2019.
5. Beniamino Accattoli and Delia Kesner. The structural λ-calculus. *Int. Conf. on Computer Science Logic* (CSL), Brno, Czech Republic, LNCS 6247, pages 381–395. Springer, 2010.
6. Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. *Int. Symposium on Functional and Logic Programming* (FLOPS), Kobe, Japan, LNCS 7294, pages 4–16. Springer, 2012.
7. Hendrik P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*, volume 103. North Holland, revised edition, 1984.
8. Erika De Benedetti and Simona Ronchi Della Rocca. A type assignment for λ-calculus complete both for FPTIME and strong normalization. *Information and Computation*, 248:195–214, 2016.
9. Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science*, 9(4), 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. *Annals of Pure Applied Logic*, 109(3):205–241, 2001.
12. Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The Bang Calculus Revisited. Extended report, 2020. https://arxiv.org/abs/2002.04011.
13. 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.
14. Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. *Int. Conf. on Foundations of Software Science and Computation Structures* (FOSSACS), Grenoble, France, LNCS 8412, pages 103–118. Springer, 2014.
15. Jules Chouquet and Christine Tasson. Taylor expansion for Call-By-Push-Value. *Int. Conf. on Computer Science Logic* (CSL), Barcelona, Madrid, *LIPIcs* 152, 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 20120.
16. Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Archive for Mathematical Logic*, 19(1):139–156, 1978.
17. Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.
18. Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul.* PhD thesis, Université Aix-Marseille II, 2007.

19. Daniel de Carvalho. The relational model is injective for multiplicative exponential linear logic. *Int. Conf. on Computer Science Logic* (CSL), Marseille, France, *LIPIcs* 62, pages 41:1–41:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
20. Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, 28(7):1169–1203, 2018.
21. Daniel de Carvalho and Lorenzo Tortora de Falco. A semantic account of strong normalization in linear logic. *Information and Computation*, 248:104–129, 2016.
22. Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. A semantic measure of the execution time in linear logic. *Theoretical Computer Science*, 412(20):1884–1902, 2011.
23. Vincent Danos. La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du lambda-calcul). PhD thesis, Université Paris 7, 1990.
24. Thomas Ehrhard. Collapsing non-idempotent intersection types. *Int. Conf. on Computer Science Logic* (CSL), Fontainebleau, France, *LIPIcs* 16, pages 259–273. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
25. Thomas Ehrhard. Call-by-push-value from a linear logic point of view. *European Symposium on Programming* (ESOP), Eindhoven, The Netherlands, LNCS 9632, pages 202–228. Springer, 2016.
26. Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. *Int. Symposium on Principles and Practice of Declarative Programming* (PPDP), Edinburgh, United Kingdom, pages 174–187. ACM, 2016.
27. Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *TCS*, 403(2-3):347–372, 2008.
28. Philippa Gardner. Discovering needed reductions using type theory. *Int. Conf. on Theoretical Aspects of Computer Software* (TACS), Sendai, Japan, LNCS 789, pages 555–574. Springer, 1994.
29. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
30. Jean-Yves Girard. Normal functors, power series and λ-calculus. *Annals of Pure Applied Logic*, 37(2):129–177, 1988.
31. Giulio Guerrieri. Towards a semantic measure of the execution time in call-by-value lambda-calculus. *Joint Int. Workshops on Developments in Computational Models and Intersection Types and Related Systems* (DCM/ITRS), Oxford, UK, *EPTCS* 283, pages 57–72, 2018.
32. Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two girard's translations. *Joint Int. Workshops on Linearity & Trends in Linear Logic and Applications* (Linearity-TLLA), Oxford, UK, EPTCS, pages 15–30, 2019.
33. Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. Computing connected proof(-structure)s from their taylor expansion. *Int. Conf. on Formal Structures for Computation and Deduction* (FSCD), Porto, Portugal, *LIPIcs* 52, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
34. Delia Kesner. Reasoning about call-by-need by means of types. *Int. Conf. on Foundations of Software Science and Computation Structures* (FOSSACS), Eindhoven, The Netherlands, LNCS 9634, pages 424–441. Springer, 2016.
35. Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. *Int. Conf. on Theoretical Computer Science* (IFIP/TCS), Rome, Italy, LNCS 8705, pages 296–310. Springer, 2014.

36. Delia Kesner and Pierre Vial. Types as resources for classical natural deduction. *Int. Conf. on Formal Structures for Computation and Deduction* (FSCD), Oxford, UK, *LIPIcs* 84, pages 24:1–24:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
37. Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation.* Springer, 2004.
38. 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.
39. John Maraist and Martin Odersky and David N. Turner and Philip Wadler. Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus. Theoretical Computer Science 228(1-2), pp. 175–210, 1999.
40. Michele Pagani and Simona Ronchi Della Rocca. Solvability in resource lambda-calculus. *Int. Conf. on Foundations of Software Science and Computational Structures* (FOSSACS), Paphos, Cyprus, LNCS 6014, pages 358–373. Springer, 2010.
41. Laurent Regnier. Une équivalence sur les lambda-termes. *TCS*, 126(2):281–292, 1994.
42. José Espírito Santo, Luís Pinto, and Tarmo Uustalu. Modal embeddings and calling paradigms. *Int. Conf. on Formal Structures for Computation and Deduction* (FSCD), Dortmund, Germany, *LIPIcs* 131, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.