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


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 them1. 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, 35, 14, 20, 33, 23, 22, 13, 44].

In this satisfactory enough? A major observation concerning $\beta$-reduction in $\lambda$-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 $\Phi$ 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 $\Phi$ 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 λ!-calculus [12], based on the bang calculus introduced in [26], which in turn extends ideas by T. Ehrhard [25]. The granularity of the λ!-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 \(\Phi\) 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 λ! 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 paper 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 \((T_\varepsilon)\), values and contexts:

- **(Terms)** \( t, u, r \ ::= v \mid t \ u \mid t[\text{x}\backslash u] \)
- **(Values)** \( v ::= x \in \mathcal{X} \mid \lambda x.t \)
- **(Contexts)** \( C ::= \emptyset \mid \emptyset L \)
- **(List Contexts)** \( L ::= \emptyset \mid \emptyset L(\text{x}\backslash t) \)
- **(CBN Contexts)** \( N ::= \emptyset \mid \emptyset t \mid \lambda x.N \mid N[\text{x}\backslash u] \)
- **(CBV Contexts)** \( V ::= \emptyset \mid \emptyset V \mid t V \mid V[\text{x}\backslash u] \mid t[\text{x}\backslash v] \)

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

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:

- (distant Beta) \( L(\lambda x.t) \ u \rightarrow_{\text{dB}} L(t[\text{x}\backslash u]) \)
- (substitute term) \( t[\text{x}\backslash u] \rightarrow_{\text{sn}} t \{\text{x}\backslash u\} \)
- (substitute value) \( t[\text{x}\backslash L(v)] \rightarrow_{\text{sv}} L(t \{\text{x}\backslash v\}) \)

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

\[
\rightarrow_{n} := N(\rightarrow_{\text{dB}} \cup \rightarrow_{\text{sn}}) \quad \text{and} \quad \rightarrow_{v} := V(\rightarrow_{\text{dB}} \cup \rightarrow_{\text{sv}})
\]

The resulting CBN/CBV formulations are now based on distinguished multiplicative (cf. \( \text{dB} \)) and exponential (cf. \( \text{sn} \) and \( \text{sv} \)) steps, called resp. \( m \)-steps and \( e \)-steps, thus inheriting the nature of cut elimination rules in LL. Notice that the number of \( m \) and \( e \)-steps in a
normalization sequence is not always the same: e.g. \( x[y\cdot x] \rightarrow_n y \) has only one \( e \)-step, and 
(\(\lambda x.x\)) (\(z I\)) \( \rightarrow_r x[z\cdot x I] \) has only one \( m \)-step. We write \( t \not\rightarrow_n \) (resp. \( t \not\rightarrow_r \)), and call \( t \) an \( n \)-normal form (resp. \( v \)-normal form), if \( t \) cannot be reduced by means of \( \rightarrow_n \) (resp. \( \rightarrow_r \)).

Both CBN and CBV are non-deterministic: \( t \rightarrow_n s \) and \( t \rightarrow_n s \) 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 \( n \)/\( v \)-normal forms can be alternatively characterised by the following grammars \[12\]:

\[
\begin{align*}
(CBN \text{ Neutral}) \quad & ne_a ::= x \in \mathcal{X} \mid ne_a t \\
(CBN \text{ Normal}) \quad & no_a ::= \lambda x.no_a \mid ne_a
\end{align*}
\]

\[
\begin{align*}
(CBV \text{ Variable}) \quad & vr_r ::= x \in \mathcal{X} \mid vr_r[x\cdot ne_a] \\
(CBV \text{ Neutral}) \quad & ne_r ::= vr_r, no_r \mid ne_r, no_r \mid ne_r[x\cdot ne_a] \\
(CBV \text{ Normal}) \quad & no_r ::= \lambda x.t \mid vr_r \mid ne_r \mid no_r[x\cdot ne_a]
\end{align*}
\]

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 \( \lambda \)-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 \( \rightarrow_{dn} \) is a deterministic version of \( \rightarrow_n \) defined as:

\[
\begin{align*}
\frac{L(\lambda x.t) u \rightarrow_{dn} L(t[x\cdot u])}{t[x\cdot u] \rightarrow_{dn} t[x\cdot u]} & \quad \frac{t \rightarrow_{dn} s \text{ and } \neg abs(t)}{t u \rightarrow_{dn} s u} & \quad \frac{t \rightarrow_{dn} s}{\lambda x.t \rightarrow_{dn} \lambda x.u}
\end{align*}
\]

Similarly, the reduction relation \( \rightarrow_{dv} \) is a deterministic version of \( \rightarrow_v \) defined as:

\[
\begin{align*}
\frac{L(\lambda x.t) u \rightarrow_{dv} L(t[x\cdot u])}{t[x\cdot L(v)] \rightarrow_{dv} L(t[x\cdot v])} & \quad \frac{t \rightarrow_{dv} s \text{ and } \neg abs(t)}{t u \rightarrow_{dv} s u} & \quad \frac{t \rightarrow_{dv} s \text{ and } \neg abs(t)}{t u \rightarrow_{dv} s u}
\end{align*}
\]

As a matter of notation, for \( \mathcal{X} \in \{dn,dv\} \), we write \( t \rightarrow_{n,c}^{(m,c)} u \) if \( t \rightarrow_{n,c} u \) using \( m \) multiplicative steps and \( c \) 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 \( no_a/no_v \) [12].

> Proposition 1. Let \( t \in T_{\mathcal{X}} \). Then, \( t \not\rightarrow_a \) iff \( t \not\rightarrow_{dn} \) iff \( t \in no_a \) and \( t \not\rightarrow_r \) iff \( t \not\rightarrow_{dv} \) iff \( t \in no_r \).

(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 \( n\)-size (resp. \( v\)-size) function:

\[
\begin{align*}
|\lambda x.t|_n & := |\lambda x.t|_n + 1 & |t.u|_n & := |t|_n + 1 & |t[x\cdot u]|_n & := |t|_n \\
|\lambda x.t|_v & := 0 & |t.u|_v & := |t|_v + |u|_v + 1 & |t[x\cdot u]|_v & := |t|_v + |u|_v
\end{align*}
\]
3 Tight Call-by-Name

We now introduce a tight type system $\mathcal{N}$ for CBN which captures independent exact measures for $\rightarrow_\lambda$-reduction sequences. This result is not surprising, since it extends the one in [1] from the pure $\lambda$-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: $a$ types terms whose normal form is an abstraction, and $n$ types terms whose normal form is CBN neutral. The grammar of types is given by:

\[
\begin{align*}
(Tight Types) & \quad tt ::= n \mid a \\
(Types) & \quad \sigma, \tau ::= tt \mid M \mid M \rightarrow \\
(Multitypes) & \quad M, N ::= [\sigma]_{i \in I} \text{ where } I \text{ is a finite set}
\end{align*}
\]

Multitypes are multisets of types. The empty multitype is denoted by $[]$, $\sqcup$ denotes multitype union, and $\subseteq$ multitype inclusion. Also, $|M|$ denotes the size of the multitype, thus if $M = [\sigma]_{i \in I}$ then $|M| = \#(I)$. 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.

Typing contexts (or just contexts), written $\Gamma, \Delta$, are functions from variables to multitypes, assigning the empty multitype to all but a finite set of variables. The domain of $\Gamma$ is given by $\text{dom}(\Gamma) \triangleq \{ x \mid \Gamma(x) \neq [] \}$. The union of contexts, written $\Gamma + \Delta$, is defined by $(\Gamma + \Delta)(x) \triangleq \Gamma(x) \sqcup \Delta(x)$. 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 $\Gamma \backslash x$ for the context $(\Gamma \backslash x)(x) = []$ and $(\Gamma \backslash x)(y) = \Gamma(y)$ if $y \neq x$.

Type judgements have the form $\Gamma \vdash^{(m,e,s)} t : \sigma$, where $\Gamma$ is a typing context, $t$ is a term, $\sigma$ is a type, and the counters $(m, e, s)$ are expected to provide the following information: $m$ (resp. $e$) indicates the number of multiplicative $n$-steps (resp. exponential $e$-steps) to normal form, while $s$ indicates the $n$-size of this normal form. It is worth noticing that the $\lambda$-calculus hides both multiplicative and exponential steps in one single $\beta$-reduction rule [1], so that only two counters suffice, one for the number of $\beta$-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 $\lambda$-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 $n$-reduction. For instance, in $KI\Omega$ the two abstractions of $K$ are consuming, while the abstraction of $I$ is persistent, and all the other constructors are also consuming, except those of $\Omega$ 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 counter slightly.

---

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 \((\text{app}^N)\) types a consuming application, meaning that its left-hand side subterm reduces to an abstraction, then causing a multiplicative step (rule dB) \(^3\) followed later by an exponential step; while

2. rule \((\text{es}^N)\) 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.

\[
\frac{\Gamma \vdash (m,e,s) \, t : n}{\Gamma \vdash (m,e,s+1) \, u : n \quad (\text{app}^N)} \quad \frac{\Gamma \vdash (m,e,s) \, t : \text{tt} \quad \text{tight}(\Gamma(x))}{\Gamma \downarrow \quad x \vdash (m,e,s+1) \lambda x. t : a \quad (\text{abs}^N)}
\]

**Figure 1** System \(N\) for the Call-by-Name Calculus: Persistent Typing Rules.

\[
\frac{\Gamma \vdash (m,e,s) \, t : \sigma \quad ([\text{var}]_e \quad x : \sigma)}{\Gamma \vdash (m,e,s) \, x : \sigma \quad (\text{var}_e)} \quad \frac{\Gamma \vdash (m,e,s) \, t : \tau \quad \text{abs}_e \quad \lambda x. t : \Gamma(x) \rightarrow \tau}{\Gamma \downarrow \quad x : (m,e,s) \, t : \tau \quad (\text{abs}_e)} \quad \frac{\Gamma \vdash (m,e,s) \, t : \sigma \quad \Delta_i \vdash (m_i,e_i,s_i) \, u : \sigma_i \quad i \in I}{\Gamma \vdash \Delta_1 \vdash (m_1,e_1,s_1) \, t : \tau \quad (\text{app}^N)} \quad \frac{\Gamma \vdash (m,e,s) \, t : \tau \quad \Delta_i \vdash (m_i,e_i,s_i) \, u : \sigma_i \quad i \in I}{(\Gamma \downarrow x) + \Delta_1 \vdash (m_1,e_1,s_1) \, t[x\backslash u] : \tau \quad (\text{es}^N)}
\]

**Figure 2** System \(N\) for the Call-by-Name Calculus: Consuming Typing Rules.

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

A multitype \([\sigma_i]_{i \in I}\) is tight, written \(\text{tight}([\sigma_i]_{i \in I})\), if \(\sigma_i \in \text{tt}\) for all \(i \in I\). A context \(\Gamma\) is said to be tight if it assigns tight multitypes to all variables. A type derivation \(\Phi \triangleright \Gamma \vdash (m,e,s) \, t : \sigma\) is tight if \(\Gamma\) is tight and \(\sigma \in \text{tt}\).

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 \(N\) is now counting separately the multiplicative and exponential steps of \(\rightarrow_n\)-reductions to normal-form.

\(^{3}\) In both [1] and [36], it is the consuming abstraction which contributes to the multiplicative steps.
27:8 Encoding Tight Typing in a Unified Framework

Theorem 2 (Soundness and Completeness).

If \( \Phi \vdash_N \Gamma \vdash (m,c) t : \sigma \) is tight, then there exists \( p \) such that \( p \in \text{no}_n \) and \( t \rightarrow_n^{(m,c)} p \) with \( m \) \( m \)-steps, \( c \) \( e \)-steps, and \( |p|_n = s \).

If \( t \rightarrow_n^{(m,c)} p \) with \( p \in \text{no}_n \), then there exists a tight type derivation \( \Phi \vdash_N \Gamma \vdash (m,c,p[s]) t : \sigma \).

Notice that the previous theorem is stated by using the general notion of reduction \( \rightarrow_n \), but the proofs are done using the deterministic reduction \( \rightarrow_{dn} \). 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 \( t_0 = K(z \ I) \ (I \ I) \) that \( \rightarrow_{\nu}-\)reduces in 2 \( m \)-steps and 2 \( e \)-steps to the term \( z \ I \in \text{no}_n \), whose \( n \)-size is 1:

\[
t_0 = K(z \ I) \ (I \ I) \rightarrow_{db} (\lambda y.x)[x\ z \ I] \ (I \ I) \rightarrow_{db} x[y\ \ I \ I][x\ z \ I] \rightarrow_{sn} (z \ I)[y\ I \ I] \rightarrow_{sn} z \ I
\]

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

\[
\begin{align*}
&x : [n] \vdash (0,0,0) x : n \quad \text{(var)} \\
&x : [n] \vdash (0,0,0) \lambda y.x : [] \rightarrow n \quad \text{(abs)} \\
&\vdash (0,0,0) K : [n] \rightarrow [] \rightarrow n \quad \text{(abs)} \\
&z : [n] \vdash (0,0,0) z : n \quad \text{(app)} \\
&z : [n] \vdash (0,0,1) z : I : n \quad \text{(app)} \\
&z : [n] \vdash (1,1,1) K(z \ I) : [] \rightarrow n \quad \text{(app)} \\
&z : [n] \vdash (2,2,1) K(z \ I) (I \ I) : n \quad \text{(app)}
\end{align*}
\]

4 Tight Call-by-Value

In this section we introduce a tight system \( V \) for CBV which captures exact measures for \( \rightarrow_{\nu}-\)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 \( V \) is given by:

\[
\begin{align*}
\text{(Tight Types)} & \quad \text{tt} & ::= & \ n \ | \ \text{v1} \ | \ \text{vr} \\
\text{(Types)} & \quad \sigma, \tau & ::= & \text{tt} \ | \ M \ | \ M \rightarrow \sigma \\
\text{(Multitypes)} & \quad M, N & ::= & [\sigma]_{i \in I} \text{ where } I \text{ is a finite set}
\end{align*}
\]

Indeed, apart from the constant \( n \) also used in CBN, we now introduce constants \( v1 \) and \( vr \), 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 \( a \) from CBN, since abstractions are in particular values, so they will be typed with \( v1 \). 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 \( V \) 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 \( \text{tt}_0 \) we write \( \overline{\text{tt}}_0 \) to denote a tight type different from \( \text{tt}_0 \). Thus for instance, \( \overline{\text{v1}} \in \{v1, n\} \).
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: (\text{var}^c_p) typing variables that will persist as
the head of a neutral term; (\text{val}^c_p) typing variable values that may be substituted by other
values, \textit{i.e.} they are placeholders for future persistent values; and (\text{var}^v_p) typing variable
values that, in particular, may be consumed as arguments. Consuming values are always
typed with multitypes. Rule (\text{app}^v_p) types neutral applications, \textit{i.e.} the left premise has type
\text{vr} or \text{n}. Rule (\text{abs}^v_p) increases the second counter, just like (\text{var}^v_p), typing a value that is
consumed as an argument. Rules (\text{app}^v_p) and (\text{app}^c_p) increment the first counter because the
(consuming) application will be used to perform a 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 (\text{var}^v_p) and (\text{abs}^v_p)), 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 (\text{app}^v_p) and (\text{app}^c_p) can also be understood by means of the subtle translation from
CBV to the \text{L}-calculus that we introduce in Sec. 7. Rule (\text{app}^c_p) is particularly useful to
type dB-redexes whose reduction does not create an exponential redex, because the argument
of the substitution created by the dB-step does not reduce to a value.

In spite of the decrements in rules (\text{app}^v_p) and (\text{app}^c_p), the counters are positive.
Lemma 4. If $\Phi \vdash \Gamma \vdash (m,c,s) t : \sigma$ then $e \geq 0$. Moreover, if $\sigma$ 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 $\Phi \vdash \Gamma \vdash (m,c,s) t : \sigma$ such that $\Gamma$ is tight.
1. If $t \in ne_v$ or $m = c = 0$, then $\sigma \in \mathbb{tt}$.
2. If $\sigma = [M \rightarrow \tau]$, then $t \not\in \mathbb{vt}_v$.

Lemma 6 (Exact Subject Reduction). Let $\Phi \vdash \Gamma \vdash (m,c,s) t : \sigma$ be a tight derivation. If $t \rightarrow_{av} t'$, then there is $\Phi' \vdash \Gamma \vdash (m',c',s) t' : \sigma$ such that

1. $m' = m - 1$ and $c' = e$ if $t \rightarrow_{av} t'$ is an m-step;
2. $c' = e - 1$ and $m' = m$ if $t \rightarrow_{av} t'$ is an 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 $x[x\setminus I] \rightarrow_{av} I$. The terms on the left and right hand side can be resp. tightly typed by the following derivations (we omit the counters):

\[
\begin{align*}
\vdash x : \mathbb{vl} & \quad (\mathsf{val}_p) \\
\vdash I : [] & \quad (\mathsf{abs}^V_p) \\
\vdash x[x\setminus I] : \mathbb{vl} & \quad (\mathsf{es}_c) \\
\vdash I : \mathbb{vl} & \quad (\mathsf{abs}^V_p)
\end{align*}
\]

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

Theorem 7 (Soundness). If $\Phi \vdash \Gamma \vdash (m,c,s) t : \sigma$ is tight, then there exists $p \in \mathbb{no}_v$ and $t \rightarrow_{av} p$ with $m$ m-steps, $e$ e-steps, and $|p|_v = s$.

As in CBN, the previous theorem is stated by using the general notion of reduction $\rightarrow_{av}$, but the proofs (notably Lem. 6) are done using the deterministic reduction $\rightarrow_{av}$. 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 $t \in \mathbb{no}_v$, then there is a tight derivation $\Phi \vdash \Gamma \vdash (0,0,|t|) t : \sigma$.

Lemma 9 (Exact Subject Expansion). Let $\Phi' \vdash \Gamma \vdash (m',c',s) t' : \sigma$ be a tight derivation. If $t \rightarrow_{av} t'$, then there is $\Phi \vdash \Gamma \vdash (m,c,s) t : \sigma$ such that

1. $m' = m - 1$ and $c' = e$ if $t \rightarrow_{av} t'$ is an m-step;
2. $c' = e - 1$ and $m' = m$ if $t \rightarrow_{av} t'$ is an e-step.

Notice that tight derivations properly counts, separately, m-steps and e-steps. As a consequence, completeness follows.
Theorem 10 (Completeness). If \( \Gamma \vdash v^{(m,e)} \) with \( p \in \text{no}_\sigma \), then there exists a tight type derivation \( \Phi \vdash \Gamma \vdash v^{(m,e)} \omega_\sigma \) of \( t : \sigma \).

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

\[
\begin{align*}
t_0 & = K(z \, I) \, (I \, I) \rightarrow_m K(x[w \, \lambda \, w \, I] \, I) \rightarrow_m (\lambda y. x[w \, \lambda \, w \, I] \, I)(I \, I) \rightarrow_m x[y \, I \, I] \, \lambda \, z \, I \\
& \rightarrow_m x[y \, I \, I] \, \lambda \, z \, I \rightarrow_m x[y \, I \, I] \, \lambda \, z \, I
\end{align*}
\]

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

5 The \( \lambda! \)-Calculus

This section briefly presents the bang calculus at a distance [12], called \( \lambda! \)-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 \( \lambda \)-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 \( \sigma \)-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 \( \lambda! \)-calculus [12] fixes these two problems at the same time. Indeed, the syntax of the bang calculus is enriched with explicit substitutions [ES], and \( \sigma \)-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 \( T \)) and contexts:

\[
\begin{align*}
(Terms) & \quad t, u & ::= & \ x \in \mathcal{X} \ | \ t \ u \ | \ \lambda x. t \ | \ ! t \ | \ \text{der} \ t \ | \ t[x \ u] \\
(List \ contexts) & \quad L & ::= & \ ! | \ L[x \ t] \\
(Surface \ contexts) & \quad S & ::= & \ ! | \ S \ t \ | \ t \ S \ | \ \lambda x. S \ | \ \text{der} \ S \ | \ S[x \ u] \ | \ t[x \ S]
\end{align*}
\]

Special terms are \( \Delta_1 = \lambda x. x \ ! x \), and \( \Delta_0 = \Delta_1 \ ! \Delta_1 \). Surface contexts do not allow the symbol \( ! \) to occur inside the bang constructor \( ! \). This is similar to weak contexts in \( \lambda \)-calculus, where \( ! \) cannot occur inside \( \lambda \)-abstractions. As we will see in Sec. 7, surface reduction in the \( \lambda! \)-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:

\[
\begin{align*}
|x|_f & := 0 & |\text{der} \ t|_f & := |t|_f & |t[x \ u]|_f & := |t|_f + |u|_f \\
|t|_f & := 0 & |\lambda x. t|_f & := 1 + |t|_f & |t \ u|_f & := 1 + |t|_f + |u|_f
\end{align*}
\]

The \( \lambda! \)-calculus is given by the set of terms \( \mathcal{T} \) and the (surface) reduction relation \( \rightarrow_f \), which is defined as the union of \( \rightarrow_m \), \( \rightarrow_e \) (substitute bang) and \( \rightarrow_d \) (distant bang), defined respectively as the closure by contexts \( S \) of the following three rewriting rules:

\[
\begin{align*}
L(\lambda x. t) \, u & \rightarrow_m L(t[x \ u]) \\
L(t[x \ L(\! u)]) & \rightarrow_d L(t[x \ u]) \\
\text{der} (L(\! t)) & \rightarrow_d L(t)
\end{align*}
\]
The rules are defined at a distance, as in CBN/CBV, in the sense that the list context \( L \) 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 \( ! \) can be decomposed in two different rules \( t[x]!u \mapsto t \{x\}u \) and \( t[x]!L(u) \mapsto L(t[x]!u) \), while \( d! \) can be decomposed in \( \text{der}(!t) \mapsto t \) and \( \text{der}(L(!t)) \mapsto L(\text{der}(!t)) \). We write \( \rightarrow_t \) for the reflexive-transitive closure of \( \rightarrow_t \). Given the translation of the bang calculus into LL proof-nets \([25]\), we refer to dB-steps as \( \text{multiplicative} \) and \( !/d! \)-steps as \( \text{exponential} \) steps. We write \( t \rightarrow_t^{(m,c)} u \) if \( t \rightarrow_t u \) using \( m \) \( m \)-steps and \( c \) \( e \)-steps.

\[\begin{align*}
\text{Example 12.} & \quad \text{Consider the following reduction sequence from } t' = K(!((z!1)!)!(1!1)): \\
 t' = K(!((z!1)!)!(1!1)) & \quad \rightarrow_{dB} (\lambda y.x)[x]!(z!1)!(1!1) \quad \rightarrow_{dB} x[y]!(1!1)[x]!(z!1) \\
 & \quad \rightarrow_{!} (z!1)[y]!(1!1) \quad \rightarrow_{!} z!1
\end{align*}\]

Notice that the second dB-step uses action at a distance, where \( L \) is \( \square[x]!(z!1) \).

The relation \( \rightarrow_t \) 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.

\[\text{Theorem 13 (Confluence [12]).} \quad \text{The reduction relation } \rightarrow_t \text{ is confluent. Moreover, any two different } \rightarrow_t \text{-reduction paths to normal form have the same length.}\]

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 \( f \)-normal form.

Neutral, Normal, and Clash-Free Terms. A term is said to be \( f \)-normal if there is no \( t' \) such that \( t \rightarrow_t t' \), in which case we write \( t \not\rightarrow_t \). However, some ill-formed \( f \)-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: \( L(!t)u \), \( t[y]L(\lambda x.u) \), or \( t(L(\lambda x.u)) \). Remark that in the three first kind of clashes, replacing \( \lambda x \) by \( ! \), and inversely, creates a (root) redex, namely \( (L(\lambda x.t))u \), \( t[x]L(!t) \) and \( L(\text{der}(!t)) \), respectively.

A term is clash free if it does not reduce to a term containing a clash, it is surface clash free, written scf, if it does not reduce to a term containing a clash outside the scope of any constructor \( ! \). Thus, \( t \) is not scf if and only if there exist a surface context \( S \) and a clash \( c \) such that \( t \rightarrow_t S(c) \). Surface clash free normal terms can be characterised as follows:

\[\begin{align*}
(\text{Neutral scf}) & \quad n_{\text{scf}} : = x \in X | n_{\text{scf}} n_{\text{scf}} | \text{der}(n_{\text{scf}}) | n_{\text{scf}}[x]n_{\text{scf}} \\
(\text{Neutral-Abs scf}) & \quad n_{\text{abs}} : = !t | n_{\text{scf}} | n_{\text{abs}}[x]n_{\text{scf}} \\
(\text{Neutral-Bang scf}) & \quad n_{\text{bang}} : = n_{\text{scf}} | \lambda x.n_{\text{scf}} | n_{\text{bang}}[x]n_{\text{scf}} \\
(\text{Normal scf}) & \quad n_{\text{scf}} : = n_{\text{scf}} | n_{\text{abs}} | n_{\text{bang}}
\end{align*}\]

\[\text{Proposition 14 (Clash-Free Normal Terms [12]).} \quad \text{Let } t \in T. \quad \text{Then } t \text{ is a surface clash free } f \text{-normal term iff } t \in n_{\text{scf}}.\]

## 6 A Tight Type System for the \( \lambda! \)-Calculus

The methodology used to define the type system \( B \) 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 \( \lambda \)-calculus. In the case of the
\( \lambda \)-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:

\(\begin{align*}
\text{(Types)} & \quad \sigma, \tau : \sigma \text{, } \tau, \text{ resp. } M \text{, } M \to \sigma \\
\text{(Multitype)} & \quad \mathcal{M}, \mathcal{N} : [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set} \\
\text{(Tight Types)} & \quad \texttt{tt} ::= n \mid a \mid \texttt{vl}
\end{align*}\)

The constant \( a \) (resp. \( \texttt{vl} \)) types whose normal form has the shape \( L(\lambda x.t) \) (resp. \( L(!t) \), \( i.e. \) values in the \( \lambda \)-calculus sense), and the constant \( n \) types terms whose normal form is a neutral \( \texttt{acf} \). 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).

\[
\begin{align*}
\Gamma \vdash (m, e, s) t : (\sigma_1, \sigma_2) & \quad \Delta \vdash (m', e', s') u : \bar{s} \\
\Gamma + \Delta \vdash (m + m', e + e', s + s') t u : n & \quad (\text{app}_p) \\
\Gamma \vdash (m, e, s) t : \texttt{tt} & \quad \text{tight}(\Gamma(x)) \quad (\text{abs}_p) \\
\Gamma \not\vdash (m, e, s) & \quad \Gamma \not\vdash (m, e, s) \quad (\text{bg}_p)
\end{align*}
\]

\( \text{Figure 5} \) System \( \mathcal{B} \) for the \( \lambda \)-Calculus: Persistent Typing Rules.

\[
\begin{align*}
\Gamma \vdash (m, e, s) t : \mathcal{M} & \quad \tau \quad \Delta \vdash (m', e', s') u : \texttt{n} \quad \text{tight}(\mathcal{M}) \\
\Gamma + \Delta \vdash (m + m', e + e', s + s') t u : \tau & \quad (\text{app}_c) \\
\Gamma \vdash (m, e, s) t : \mathcal{M} & \quad \tau \quad \Delta \vdash (m', e', s') u : \texttt{n} \quad \text{tight}(\mathcal{M}) \\
\Gamma + \Delta \vdash (m + m', e + e', s + s') t u : \tau & \quad (\text{app}_c) \\
\Gamma \not\vdash (m, e, s) & \quad \Gamma \not\vdash (m, e, s) \quad (\text{bg}_c)
\end{align*}
\]

\( \text{Figure 6} \) System \( \mathcal{B} \) for the \( \lambda \)-Calculus: Consuming Typing Rules.

As in CBV, the \( \texttt{s}! \) 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 (\text{dr}_c) does not count der constructors, according to the definition of \( \llbracket t \rrbracket_\epsilon \) 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,

\textbf{Theorem 15 (Soundness and Completeness).} 
1. If \( \Phi \triangleright_\mathcal{B} \Gamma \vdash (m, e, s) t : \sigma \) \( \text{tight} \), then there exists \( p \) such that \( p \in \text{no}_\text{esc} \) and \( t \rightarrow^*_\mathcal{B} (m, e, s) p \) with \( m \) \( \text{m} \)-steps, \( e \) \( \text{e} \)-steps, and \( |p|_\lambda = s \).
2. If \( t \rightarrow^*_\mathcal{B} (m, e, s) p \) with \( p \in \text{no}_\text{esc} \), then there exists a tight type derivation \( \Phi \triangleright_\mathcal{B} \Gamma \vdash (m, e, |p|) t : \sigma \).
Example 16. Consider $t'_0 = K(\lambda (z ! I) z (\lambda (I ! I) I))$ from Ex. 12, which normalises in 2 m-steps and 2 e-steps to $z ! I \in \omega_{\text{ref}}$ of f-size 1. A tight derivation for $t'_0$ with appropriate final counterm $\langle 2, 2, 1 \rangle$ is given below.

\[
\frac{x : [n] \vdash (0,0,0) x : n}{(\text{var})} \quad \frac{z : [n] \vdash (0,0,0) z : n}{(\text{var})} \quad \frac{z : [n] \vdash (0,0,0) ! I : : 1}{(b_{gb})} \quad \frac{\vdash \langle 0,0,0 \rangle (\text{app})}
\]

Notice that the only persistent rules are $(b_{gb})$ and $(\text{app})$, used to type $z ! I$. Indeed, $z ! I$ is the 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), $cbn(t)$ reduces to $cbv(u)$ (resp. $cbv(t)$ reduces to $cbv(u)$) 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 CBN and CBV, such that when $t$ reduces to $u$ in CBN (resp. CBV), $cbn(t)$ reduces to $cbv(u)$ (resp. $cbv(t)$ reduces to $cbv(u)$) 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 $\lambda!$-calculus, written $\_^n$ and $\_^v$ resp., are inductively defined as:

\[
\begin{align*}
\lambda x. t & \equiv \lambda x.t^n \\
(t u) & \equiv t^n ! u^n \\
(t[x/u]) & \equiv t^n[x!u^n]
\end{align*}
\]

\[
\begin{align*}
x & \equiv x \\
(\lambda x. t)^v & \equiv !x \\
(t u)^v & \equiv \begin{cases} 
L(s)u^v & \text{if } t^v = L(\lambda s) \\
\text{der}(t^v)u^v & \text{otherwise}
\end{cases}
\end{align*}
\]

Both translations extend to list contexts $L$ 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 $tu$ is usually defined as $\text{der}(t^n)u^n$ (see e.g. [26]). This definition does not preserve normal forms, i.e. $x y$ is a v-normal form but its translated version $\text{der}(\lambda x. y)$ is not a f-normal form. We restore this fundamental property by using the well-known notion of superdevelopment [10], so that $\lambda!$-reductions are applied by the translation on the fly. Moreover, simulation of CBN/CBV in the $\lambda!$-calculus also holds.

Lemma 17 (Simulation [12]). Let $t \in T_{\lambda}$.

1. $t \not\rightarrow_n$ implies $t^n \not\rightarrow_{\lambda}$, and $t \rightarrow_n s$ implies $t^n \rightarrow_{\lambda} s^n$.
2. $t \not\rightarrow_v$ implies $t^v \not\rightarrow_{\lambda}$, and $t \rightarrow_v s$ implies $t^v \rightarrow_{\lambda} s^v$.
Consider again \( t_0 = K \varepsilon 1 \varepsilon 1 \). To illustrate the CBN case (Lem. 17.1), notice that the reduction sequence \( t_0 \to_{CBV} \varepsilon z \varepsilon = s_0 \) given in Ex. 3 is translated to the sequence \( t_0^n = K (! \varepsilon 1 \varepsilon 1) (! \varepsilon 1 \varepsilon 1) t'_0 \to_{CBV} \varepsilon z \varepsilon = s_0^n \) given in Ex. 12.

To illustrate the CBV case (Lem. 17.2), consider the sequence \( t_0 \to_{CBV} x[x \varepsilon z \varepsilon 1] = s_1 \) in Ex. 11. Then for \( \Gamma' = \lambda w . \varepsilon w \) we have:

\[
\begin{align*}
t_0^n &= t \to_{CBV} (\lambda y . ! y \varepsilon x) (z \varepsilon \Gamma') (\Gamma' \varepsilon \Gamma') \\
&\to_{\eta!} (\lambda y . \varepsilon x) [x \varepsilon z \varepsilon \Gamma'] (\Gamma' \varepsilon \Gamma') \\
&\to_{\eta!} \varepsilon (\lambda y . \varepsilon x) [x \varepsilon z \varepsilon \Gamma'] \\
&\to_{\eta!} \varepsilon (\lambda y . \varepsilon x) [x \varepsilon z \varepsilon \Gamma'] = s_1^n.
\end{align*}
\]

Notice how this sequence requires extra reduction steps with respect to the one given in Ex. 11. Indeed, the \( \varepsilon \)-step \( \to_{\eta!} \) in the \( \lambda \varepsilon \)-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 \( \lambda \varepsilon \)-calculus. First we inject the set of types for \( \mathcal{N} \) (generated by the base types \( n \) and \( a \)) into the set of types of \( B \) (generated also by the base type \( \nu l \)) by means of the function: \( n^n \equiv n, a^n \equiv a, (\mathcal{M} \to \sigma)^n \equiv \mathcal{M}^n \to \sigma^n \) and \( [\sigma_i]_{i \in I} \equiv [\sigma_i^n]_{i \in I} \). Then we translate terms, using the function \( ^n \) from Sec. 7. Translation of contexts is defined as expected: \( \Gamma^n = \{ x_i : \mathcal{M}^{n_i} \}_{i \in I} \). Another notion is needed to restrict \( B \) derivations to those that come from the translation of some \( \mathcal{N} \) derivation. Indeed, a \( B \) derivation \( \Phi \) is \( n\)-relevant if all the contexts and types involved in \( \Phi \) are in the image of the translation \( ^n \). We then obtain:

\[ \text{Theorem 19.} \quad \Phi \vdash_{\mathcal{N}} \Gamma \vdash_{\mathcal{M} ; e , v ; a} t : \sigma \text{ if and only if } \Phi' \vdash_{B} \Gamma^n \vdash_{\mathcal{M} ; e , v ; a} t^n : \sigma^n \text{ is } n\text{-relevant}. \]

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

\[ \text{Corollary 20.} \quad \text{If } \Phi \vdash_{\mathcal{N}} \Gamma \vdash_{\mathcal{M} ; e , v ; a} t : \sigma \text{ is tight, then there exists } p \in \text{no}_{\text{ext}} \text{ such that } t^n \to_{\mathcal{M} ; e , v ; a} p \text{ with } m \text{-steps, } e \text{-steps, and } |p|_2 = s. \text{ Conversely, if } \Phi' \vdash_{B} \Gamma^n \vdash_{\mathcal{M} ; e , v ; a} t^n : \sigma^n \text{ is tight and } n\text{-relevant, then there exists } p \in \text{no}_{\text{ext}} \text{ such that } t \to_{\mathcal{M} ; e , v ; a} p \text{ with } m \text{-steps, } e \text{-steps, and } |p|_2 = s. \]

This result shows only from the tight type system \( \mathcal{N} \) it is possible to extract exact measures for the image of the CBV in the \( \lambda \varepsilon \)-calculus, but more interestingly, also that from the tight type system \( B \) for the \( \lambda \varepsilon \)-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 \( B \), and we need to properly translate types. To that end, we introduce two mutually dependent translations \( _v^\mathcal{T} \) and \( _v^\mathcal{V} \):

\[
\begin{align*}
tt^\mathcal{T} &\overset{\equiv}{=} \tt & & \text{if } \tt \neq \nu r \\
v^\mathcal{T} &\overset{\equiv}{=} \nu \\
(v \tau)^\mathcal{T} &\overset{\equiv}{=} \tau & & \nu r^\mathcal{T} \overset{\equiv}{=} [\nu] \\
(\mathcal{M} \to \sigma)^\mathcal{T} &\overset{\equiv}{=} \mathcal{M}^\mathcal{T} \to \sigma^\mathcal{T} & & (\mathcal{M} \to \sigma)^\mathcal{V} \overset{\equiv}{=} \mathcal{M}^\mathcal{V} \to \sigma^\mathcal{V} \\
([\sigma_i]_{i \in I})^\mathcal{T} &\overset{\equiv}{=} [\sigma_i^\mathcal{T}]_{i \in I} & & ([\sigma_i]_{i \in I})^\mathcal{V} \overset{\equiv}{=} [\sigma_i^\mathcal{V}]_{i \in I}
\end{align*}
\]

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

\[ \text{Theorem 21.} \quad \Phi \vdash_{\mathcal{N}} \Gamma \vdash_{\mathcal{M} ; e , v ; a} t : \sigma \text{ if and only if } \Phi' \vdash_{B} \Gamma^\mathcal{V} \vdash_{\mathcal{M} ; e , v ; a} t^\mathcal{V} : \sigma^\mathcal{V} \text{ is } v\text{-relevant}. \]
Encoding Tight Typing in a Unified Framework

(1) all the contexts and types involved in \( \Phi \) are in the image of the translations \( _\tau^\gamma \) and \( _\nu^\gamma \) respectively; and

(2) rule (\( \text{dr}_c \)) is only applied to terms having a type of the form \( [\mathcal{M} \rightarrow \tau] \).

To state the preservation of typing derivations between systems \( V \) and \( B \) 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 \( \text{dl} \)-redexes of the \( \lambda! \)-calculus that might be introduced when translating. For all the typing rules with two premises, we write \( \Phi_t \) and \( \Phi_u \) for the first and second premise respectively. The first measure for system \( V \) is given by induction on \( \Phi \) as follows:

(1) for (\( \text{var}_p \)), \( e(\Phi) \overset{1}{=} 1 \);

(2) for (\( \text{val}_p \)), (\( \text{abs}_p^\gamma \)) and (\( \text{var}_p^\gamma \)), \( e(\Phi) \overset{0}{=} 0 \);

(3) for (\( \text{app}_p^\gamma \)), \( e(\Phi) \overset{1}{=} e(\Phi_t) + e(\Phi_u) - 1 \) if \( \text{val}(t) \);

(4) for (\( \text{app}_p^\gamma \)) and (\( \text{appt}_p^\gamma \)), \( e(\Phi) \overset{1}{=} e(\Phi_t) + e(\Phi_u) + 1 \) if \( \neg \text{val}(t) \); and

(5) in any other case \( e(\Phi) \) is defined as the sum of the recursive calls over all premises. The second measure for system \( B \) is also defined by induction on \( \Phi \) as follows:

(1) for (\( \text{bg}_p \)) and (\( \text{var}_p \)), \( e(\Phi) \overset{0}{=} 0 \);

(2) for (\( \text{app}_p \)), \( e(\Phi) \overset{1}{=} e(\Phi_t) + e(\Phi_u) - 1 \) if \( \text{val}(t) \);

(3) for (\( \text{app}_p \)) and (\( \text{appt}_p \)), \( e(\Phi) \overset{1}{=} e(\Phi_t) + e(\Phi_u) + 1 \) if \( \neg \text{val}(t) \); and

(4) in any other case \( e(\Phi) \) is defined as the sum of the recursive calls over all premises. Then, we obtain:

\[ \text{Theorem 21.} \]

1. If \( \Phi \triangleright_{\text{V}} \Gamma \vdash (m,c,s) t : \sigma \), then \( \Phi' \triangleright_{\text{V}} \Gamma^\gamma \vdash (m,c',s') t^\gamma : \sigma^\gamma \) is \( \nu^\gamma \)-relevant with \( c' = c + e(\Phi) \).

2. If \( \Phi' \triangleright_{\text{V}} \Gamma \vdash (m,c,s) t : \sigma \) is \( \nu^\gamma \)-relevant, then \( \Phi \triangleright_{\text{V}} \Gamma \vdash (m,c,s) t : \sigma \) with \( c = c' - e(\Phi') \).

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

\[
\begin{align*}
\frac{x : [\text{vr}] \vdash (0,0,0) x : \text{vr}}{\vdash (0,1,0) \lambda x.x : [([\text{vr}] \rightarrow \text{vr})]} & \quad \frac{y : [\text{vr}] \vdash (0,1,0) y : [\text{vr}]}{\vdash (1,1,0) (\langle x, y \rangle : y : \text{vr})}
\end{align*}
\]

\[
\begin{align*}
\frac{x : [n] \vdash (0,0,0) x : n}{\vdash (0,1,0) \lambda x. !x : [n] \rightarrow [n]} & \quad \frac{y : [n] \vdash (0,0,0) y : n}{\vdash (2,0,0) (\langle \lambda x. !x \rangle ! y : [n])}
\end{align*}
\]

Notice that \( \text{tight}(\Gamma) \) if and only if \( \text{tight}(\Gamma^\gamma) \). Unfortunately, this is not sufficient to translate a tight derivation in \( V \) into a tight derivation in \( B \). Indeed, the variable \( x \) of type \( \text{vr} \) translates to \( !x \) of type \( [n] \). However, this only happens if the derived type is \( \text{vr} \), which is an auxiliary type of system \( V \) used to identify variables that are not used as values because they will be applied to some argument. In the case of \( \text{f}-\text{relevant} \) \( V \) derivations, defined as not deriving type \( \text{vr} \), 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:

\[ \text{Corollary 22.} \]

If \( \Phi \triangleright_{\text{V}} \Gamma \vdash (m,c,s) t : \sigma \) is tight and \( \text{f}-\text{relevant} \), then there exists \( p \in \text{noscf} \) such that \( t^\gamma \overset{1}{\rightarrow}_{(m,c)} p \) with \( m \) \( \text{m} \)-steps, \( e + e(\Phi) \) \( \text{e} \)-steps, and \( |p|_v = s \). Conversely, if \( \Phi' \triangleright_{\text{V}} \Gamma \vdash (m,c',s) t^\gamma : \sigma^\gamma \) is tight and \( \text{f}-\text{relevant} \), then there exists \( p \in \text{no}_s \) such that \( t \overset{1}{\rightarrow}_{(m,c)} p \) with \( m \) \( \text{m} \)-steps, \( e' - e(\Phi') \) \( \text{e} \)-steps, and \( |p|_v = s \).


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 been 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


27:20 Encoding Tight Typing in a Unified Framework


