Useful Call-by-Value: A Semantic Interpretation via Quantitative Types

Pablo Barenbaum
Universidad Nacional de Quilmes (CONICET), Buenos Aires, Argentina
Instituto de Ciencias de la Computación, UBA, Buenos Aires, Argentina

Delia Kesner
Université Paris Cité, CNRS, IRIF, France

Mariana Milicich
Université Paris Cité, CNRS, IRIF, France

Abstract

Useful evaluation is an optimised evaluation mechanism for functional programming languages. It relies on representing terms with *sharing* and imposing a restricted notion of *useful substitutions*, that intuitively disallows copying subterms that do not contribute to the progress of the computation.

In particular, *useful call-by-value evaluation* optimises the standard *call-by-value* strategy by preserving its original semantics. This preservation result has been shown by means of syntactical rewriting techniques, difficult to adapt to alternative variants of the calculi at play.

In this work, we present the first semantic model of useful call-by-value evaluation through the *non-idempotent intersection type system* $\mathcal{U}$. Our first contribution is a characterisation of termination for useful call-by-value evaluation via system $\mathcal{U}$. That is, a term is typable in system $\mathcal{U}$ if and only if it terminates in the useful call-by-value strategy. As a second contribution, we show that system $\mathcal{U}$ provides a *quantitative* interpretation for useful call-by-value evaluation, offering *exact* step-count information for program evaluation. Our third contribution is that termination in call-by-value and useful call-by-value are equivalent. This ensures in particular that call-by-value, which is (potentially) *erasing*, and useful call-by-value, which is *non-erasing*, are *observationally equivalent*. Even though the specification of the operational semantics of useful evaluation is highly complex, system $\mathcal{U}$ is notably simple. As far as we know, system $\mathcal{U}$ is one of the scarce quantitative type systems capturing exactly the substitution step-count for variables *and* abstractions in an open call-by-value strategy.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Theory of computation → Type theory; Theory of computation → Operational semantics

Keywords and phrases Lambda calculus, Evaluation strategies, Call-by-Value, Useful Evaluation, Intersection types, Quantitative models

Digital Object Identifier 10.4230/LIPIcs.CSL.2026.47

Related Version *Technical Report*: `https://arxiv.org/abs/2404.18874` [16]

Funding This work was supported by the European Union through the MSCA SE project QCOMICAL (Grant Agreement ID: 101182520).
Mariana Milicich: This project has received funding from the EU Horizon 2020 research and innovation programme MSCA No 945332.

1 Introduction

Implementing a model of computation efficiently requires designing an operational semantics that closely adheres to the original model, while optimising the key operations for computational efficiency. This creates a significant gap between the original model and its optimised version, making it crucial to guarantee that both notions are *observationally equivalent*, meaning that the observable behaviour of any program in the optimised model should be indistinguishable from its behaviour in the original one.

A typical and well-known example of this phenomenon is call-by-need (CBNeed), which can be shown to be observationally equivalent to call-by-name (CBN) [12, 41, 36]. Another recent example of an optimisation technique in the theory of functional programming languages is *useful evaluation*, introduced by Accattoli and Dal Lago [9]. This notion is based on a graph *representation of λ-terms* which implements *sharing* by means of *explicit substitutions* (ESs) (see [35] for a survey on ES). Useful evaluation aims to perform duplication only when this *contributes to the progress of the computation*, that is, to the creation of β-redexes.

Defining useful evaluation is challenging because its corresponding operational semantics must precisely capture what "contributing to the progress of the computation" means. Indeed, this makes useful evaluation *context-sensitive* by nature. In the call-by-value (CBV) framework, useful evaluation can be seen as an optimised implementation of standard CBV that effectively preserves its original semantics [1, 3]. This preservation result has been proved by means of syntactical tools based on rewriting theory.

In general, proving equivalence between two calculi through operational reasoning is inherently complex, often relying on various syntactic techniques such as sharing, unfolding, residual theory, standardisation, among others. Moreover, syntactic proofs of observational equivalence are often *ad hoc* and difficult to generalise to alternative variants of the calculi at play. In contrast, the semantic methods used in this work have provided abstract and conceptual proofs when applied before to other cases; this is for example the emblematic semantic equivalence between CBNeed and CBN [36]. Here, we extend these methods to the CBV framework to capture in particular usefulness.

Quantitative Interpretations. We rely on the *non-idempotent* flavour of *intersection types* (IT) to provide a model of useful CBV evaluation. IT extend simple types with an intersection type constructor $\cap$, allowing a program t to be typed with $\alpha \cap \beta$ if t types with both α and β independently. Initially introduced as *models* to capture *qualitative* computational properties of functional programs [23, 24], intersection type systems (ITS) are able in particular to characterise termination of evaluation strategies: a program t terminates in a given evaluation strategy if and only if t is typable in an associated type system.

In the original formulation of IT, the type constructor $\cap$ is considered to be **idempotent**, meaning that $\sigma \cap \sigma = \sigma$. So, an intersection $\alpha_1 \cap \ldots \cap \alpha_n$ can be represented as the *set* $\{\alpha_1, \ldots, \alpha_n\}$. More recent works have adopted a **non-idempotent** intersection type constructor [30, 22, 25]. In this setting, an intersection $\alpha_1 \cap \ldots \cap \alpha_n$ can be represented as the *multiset* $[\alpha_1, \ldots, \alpha_n]$. Like their idempotent precursors, non-idempotent ITS still allow characterising operational properties of programs by means of typability but also yield a substantial improvement: they provide *quantitative* measures about these properties [30, 22, 25]. For instance, from a typing derivation of a program t, it is possible to extract an *upper bound* – or even an *exact measure* – for the number of steps which is necessary to reach the normal form of t.

More precisely, exact measures can be obtained in the so-called *tight* quantitative type systems [2], by decorating the type judgements with *counters*, and by imposing a *tightness* condition on typing derivations to ensure that they are *minimal*. For instance, the counter m may be used in the typing judgement $\Gamma \vdash^m t : \alpha$ to capture the fact that t evaluates in exactly m reduction steps to a normal form.

To (quantitatively) characterise termination for a given evaluation strategy in such type systems, one relies on the results of *soundness* and *completeness* of the type system with respect to the corresponding evaluation strategy. Here, *quantitative* **soundness** means that for any *tight* type derivation of a program t with counter m, the program t evaluates to

normal form in exactly m steps. Conversely, *quantitative* **completeness** means that every reduction sequence to normal form of a given length corresponds to a *tight* typing derivation with appropriate counters.

Quantitative type systems based on non-idempotent ITS have been applied to various evaluation strategies in the λ-calculus for obtaining upper bounds and exact measures [17, 21, 2, 25, 39], as well as in the contexts of classical calculi [37, 38], call-by-value [28, 4, 34, 6], call-by-need [13, 14, 7, 40], call-by-push-value [19], languages with effects [11], linear logic [27, 26], etc. These type systems can be seen as semantic interpretations, akin to *relational models* in the usual sense of linear logic [32, 18].

In the framework of Plotkin's CBV [43], non-idempotent ITS have been introduced by Ehrhard [28], and then extended to calculi with sharing (see *e.g.*, [4, 7, 19, 39, 6]). All these type systems characterise termination of the corresponding CBV calculus but their underlying notion of *tightness* does not seem to be easily adaptable to *useful* CBV. The problem of finding a semantic model of useful CBV capable of providing the *exact* number of steps to reach a normal form thus remained open.

Previous definitions of useful CBV in the literature lack semantic models, and existing quantitative models of CBV do not account for usefulness. This work aims to provide the first semantic interpretation of useful CBV evaluation by introducing a *quantitative model* that admits a *tight* extension. In particular, our work provides one of the scarce quantitative type systems capturing exactly the substitution step-count for variables and abstractions in an open CBV strategy.

Motivation and Goal of this Paper. As discussed in the previous subsection, semantic models are often *qualitative*, in that they capture *extensional* properties of programs, such as observational equivalence. In contrast, models based on non-idempotent intersection types refine this perspective by capturing *intensional* properties of programs, such as the length of reductions to normal form, and therefore considered to be *quantitative*. Qualitative semantic models are unable to distinguish between alternative implementations, such as CBN vs. CBNeed, or CBV vs. useful CBV. The motivation of this work is to **deepen the understanding** of **useful CBV evaluation** by developing tools that enable quantitative reasoning about program behaviour. The main goal is to develop a sound and complete **quantitative model** based on **non-idempotent intersection types** for useful CBV. In particular, since it is well-known that there can be an exponential gap between function application steps and substitution steps (*a.k.a.* the *size explosion problem*, see *e.g.* [9]), our goal is to include in the design of such a quantitative model a way to distinguish between these two forms of reduction steps. Such a distinction has several applications, in particular, in the analysis of cost models and the design of optimisation techniques.

Contributions and Structure of the Paper. We begin in Section 2 by recalling Accattoli and Paolini's *Value Substitution Calculus* (vsc) [10], the CBV calculus we use in this work. Whereas most functional programming languages rely on reduction for *closed* terms (*i.e.*, without free variables), reduction in the vsc applies also on *open* terms (*i.e.*, with free variables). Adopting an open CBV setting broadens the applicability of our results, as open CBV serves as a foundational framework for more complex evaluation models, like *strong* CBV [33] which is used to implement proof assistants based on dependent type theory.

Since the vsc is a (potentially) *erasing* calculus (*i.e.*, values can be erased during computation, see subsection 2.3) while useful CBV is not, relating their operational semantics is not easy. To overcome this barrier, we define a non-erasing variant of vsc we dub vsc_{ne},

which enjoys two properties: (1) termination in vsc is equivalent to termination in $\mathrm{vsc}_{\mathrm{ne}}$, and (2) termination in $\mathrm{vsc}_{\mathrm{ne}}$ is equivalent to termination in useful CBV. The first property is relatively routine (see the appendix for details), so most of the paper is devoted to studying the second one, which is challenging.

In Section 3, we define system $\mathcal{U}$, a non-idempotent intersection type system which is shown to characterise termination in $\mathrm{vsc}_{\mathrm{ne}}$. In Section 4, we recall an inductive formulation of the useful CBV evaluation strategy that has been recently defined [15]. Finally, in Section 5, we show that system $\mathcal{U}$ is the first semantic model for useful CBV evaluation, addressing the lack of a high-level, semantic characterisation of this strategy.

Our first main result is that system $\mathcal{U}$ characterises termination in both $\mathrm{vsc}_{\mathrm{ne}}$ (Theorem 3) and useful CBV (Corollary 14). Termination in $\mathrm{vsc}_{\mathrm{ne}}$ is equivalent to typability in system $\mathcal{U}$, which is in turn equivalent to termination in useful CBV. This ensures in particular that $\mathrm{vsc}_{\mathrm{ne}}$ and useful CBV are observationally equivalent. Therefore, useful CBV evaluation can be seen as an optimisation of CBV that preserves its original semantics.

As a second main result, we refine system $\mathcal{U}$ to provide *exact* quantitative information of useful CBV evaluation. Indeed, we equip type judgements of system $\mathcal{U}$ with counters that are natural numbers, and we define a notion of *tightness*, intuitively capturing minimality of derivations. The counters are meant to capture the exact step-count for program evaluation. More precisely, we show that a term t is *tightly* typable with counters m and e in system $\mathcal{U}$ if and only if t terminates in useful CBV evaluation in exactly m function application steps and e substitution steps (Corollary 14). Our contribution is novel, as previous definitions of useful evaluation (including useful CBV) lack semantic models and current quantitative interpretations of CBV do not consider *usefulness*. Notably, except for [39], which relies entirely on the *persistent/consuming* paradigm, this is the only quantitative type system for open CBV that is able to count variable *and* abstraction *substitution* steps *exactly*.

2 Reduction and Open Call-by-Value

In this section, we start by recalling some general notions of rewriting theory. We then review the *Value Substitution Calculus* (vsc), a CBV calculus introduced by Accattoli and Paolini [10], as well as a non-erasing variant of vsc called $\mathrm{vsc}_{\mathrm{ne}}$.

2.1 Preliminary Notions on Rewriting Theory

Given a **reduction system** $\mathcal{R}$, we write $\to_{\mathcal{R}}$ to denote the (one-step) reduction relation associated to $\mathcal{R}$. We write $\to_{\mathcal{R}}^{=}$ and $\to_{\mathcal{R}}^{*}$ for the reflexive and reflexive-transitive closure of $\to_{\mathcal{R}}$, respectively, and $\to_{\mathcal{R}}^{n}$ for the composition of n-steps of $\to_{\mathcal{R}}$. A term t is said to be **$\mathcal{R}$-irreducible**, or in **$\mathcal{R}$-normal form**, written $t \not\to_{\mathcal{R}}$, if there is no u such that $t \to_{\mathcal{R}} u$. A term t is said to be **$\mathcal{R}$-weakly terminating** (resp. **$\mathcal{R}$-terminating**) if there exists some $\mathcal{R}$-normal form u such that $t \to_{\mathcal{R}}^{*} u$ (resp. there is no infinite $\mathcal{R}$-sequence starting at t). A term t is **$\mathcal{R}$-diamond** (or enjoys the $\mathcal{R}$-diamond property) if $t \to_{\mathcal{R}} t_0$ and $t \to_{\mathcal{R}} t_1$, with $t_0 \neq t_1$, imply there is t' such that $t_0 \to_{\mathcal{R}} t'$ and $t_1 \to_{\mathcal{R}} t'$. A term t is **$\mathcal{R}$-confluent** if $t \to_{\mathcal{R}}^{*} t_0$ and $t \to_{\mathcal{R}}^{*} t_1$ imply there is t' such that $t_0 \to_{\mathcal{R}}^{*} t'$ and $t_1 \to_{\mathcal{R}}^{*} t'$. A relation $\mathcal{R}$ is **weakly-terminating** (resp. **terminating**, **diamond**, **confluent**) if and only if every term is $\mathcal{R}$-terminating (resp. $\mathcal{R}$-terminating, $\mathcal{R}$-diamond, $\mathcal{R}$-confluent). Any relation verifying the diamond property is in particular confluent. Moreover, if t is confluent, then its $\mathcal{R}$-normal form, if it exists, is unique [44].

2.2 The Value Substitution Calculus

It is well-known that naive extensions of Plotkin's CBV [43] to *open* CBV do not enjoy adequacy, in the sense that normal forms are not necessarily *solvable*, *i.e.* "meaningful" [1] Our starting point is the *Value Substitution Calculus* (vsc) [10], an open CBV calculus with explicit substitutions that recovers adequacy and extends Plotkin's CBV.

Given a denumerable set of **variables** $(x, y, z, \ldots)$, the syntax of the vsc is specified by:

$$t, u, s, \ldots ::= x \mid \lambda x.\, t \mid t\, u \mid t[x\backslash u] \qquad\qquad v, w, \ldots ::= x \mid \lambda x.\, t$$

describing the sets of **terms** and **values**, respectively. We also use the following notions of **substitution contexts** and **weak contexts:**

$$\mathtt{L}, \mathtt{L}', \ldots ::= \Diamond \mid \mathtt{L}[x\backslash t] \qquad\qquad \mathtt{W}, \ldots ::= \Diamond \mid \mathtt{W}\, t \mid t\, \mathtt{W} \mid \mathtt{W}[x\backslash t] \mid t[x\backslash \mathtt{W}]$$

The set of terms includes **variables**, **abstractions**, **applications**, and **closures** $t[x\backslash u]$, representing an **explicit substitution** (ES) $[x\backslash u]$ on a term t. Some recurring terms are the identity function $\mathtt{I} := \lambda x.\, x$ and the operator $\omega := \lambda x.\, x\, x$.

Free and **bound occurrences** of variables are defined as usual, where free occurrences of x in t are bound in $t[x\backslash u]$. Terms are considered up to α-renaming of bound variables. Substitution contexts are lists of ESs, and we write $t\mathtt{L}$ for the **replacement** of the hole $\Diamond$ in $\mathtt{L}$ by t, which may capture the free variables of t. For example, if $\mathtt{L} = [x\backslash y\, y][y\backslash z]$ and $t = x$, then $t\mathtt{L} = x[x\backslash y\, y][y\backslash z]$. We also consider a **meta-level operation** $t\{x := u\}$ which substitutes all the free occurrences of x by u in t. Thanks to α-renaming, we can assume that this substitution operation is *capture-free*. The sets of **free variables** of a **term** $(\mathsf{fv}(t))$ and a **context** $(\mathsf{fv}(\mathtt{L}))$ are defined as expected.

Reduction in vsc is defined by the two following rewriting rules, closed by arbitrary *weak* contexts, *i.e.*, reduction is not allowed under abstractions:

$$(\lambda x.\, t)\mathtt{L}\, u \to_{\mathsf{db}} t[x\backslash u]\mathtt{L} \qquad\qquad t[x\backslash v\mathtt{L}] \to_{\mathsf{sv}} t\{x := v\}\mathtt{L}$$

The db-rule ("distant beta") is a β-like rule that applies an abstraction $\lambda x.\, t$ to an argument u, creating an ES $[x\backslash u]$ that binds x to the argument u. This rule acts at a *distance*, meaning an arbitrary list of ESs $(\mathtt{L})$ may be in between the abstraction and its argument. For example, $(\lambda x.\, \lambda y.\, x)\, t\, u \to_{\mathsf{db}} (\lambda y.\, x)[x\backslash t]\, u \to_{\mathsf{db}} x[y\backslash u][x\backslash t]$.

The sv-rule ("substitution of values") fires an ES when x is bound to a value v surrounded by a context $\mathtt{L}$. This performs the meta-level substitution of x by v, potentially producing $n \geq 0$ copies of v, while it extrudes the substitution context $\mathtt{L}$, which must remain shared. For example, $x[y\backslash z] \to_{\mathsf{sv}} x\{y := z\} = x$ results in the erasure of the value z, while $(x\, x)[x\backslash(\lambda y.\, z)[z\backslash t]] \to_{\mathsf{sv}} ((\lambda y.\, z)\, \lambda y.\, z)[z\backslash t]$ produces two copies of $\lambda y.\, z$ but keeps a single copy of $[z\backslash t]$. Note that a term like $x[x\backslash y\, y]$ is in vsc-normal form because $y\, y$ is not a value.

The notion of **reachable variable** of a term t is sometimes relevant in vsc, due to the fact that reduction is closed by weak contexts. The set of reachable variables $\mathsf{rv}(t)$ is defined as the set of free variables that do not appear inside an abstraction. Formally:

$$\begin{array}{lcllcl} \mathsf{rv}(x) & := & \{x\} & \mathsf{rv}(\lambda x.\, t) & := & \emptyset \\ \mathsf{rv}(t\, u) & := & \mathsf{rv}(t) \cup \mathsf{rv}(u) & \mathsf{rv}(t[x\backslash u]) & := & (\mathsf{rv}(t) \setminus \{x\}) \cup \mathsf{rv}(u) \end{array}$$

[1] See [42, 29, 6] for a general discussion on CBV solvability.

2.3 The Non-Erasing Value Substitution Calculus

As part of this work, we aim to establish a correspondence between CBV and UCBV$^\bullet$, the useful CBV evaluation strategy defined in [15].

In principle, standard CBV strategies allow erasing some ESs that have become *unreachable*. For instance, $(x\,x)[y\backslash v] \to x\,x$ is an erasing step. However, erasing steps are not useful and thus UCBV$^\bullet$ does not include erasing rules. As we will see in Section 5, UCBV$^\bullet$ evaluation implements in particular a *non-erasing* variant of the vsc.

The **Non-Erasing Value Substitution Calculus** (vsc$_{\text{ne}}$) is given by the vsc syntax and the reduction is defined by the following rules, closed by arbitrary weak contexts:

$$(\lambda x.\, t)\mathtt{L}\, u \to_{\mathsf{db}} t[x\backslash u]\mathtt{L} \qquad\qquad t[x\backslash v\mathtt{L}] \to_{\mathsf{svne}} t\{x := v\}[x\backslash v]\mathtt{L} \quad \text{if } x \in \mathsf{fv}(t)$$

The svne-rule is similar to the sv-rule but requires that the variable x occurs free in t. For example, $x[y\backslash z]$ reduces to x in vsc but is already a vsc$_{\text{ne}}$-normal form. We assume the svne-rule to be capture-free, due to α-renaming ($x \notin \mathsf{fv}(v)$).

3 The Type System $\mathcal{U}$

We now present a non-idempotent intersection type system (ITS) called $\mathcal{U}$ and show that it is sound and complete for the non-erasing Value Substitution Calculus vsc$_{\text{ne}}$. The main result of this section is Theorem 3, which characterises termination in vsc$_{\text{ne}}$ by system $\mathcal{U}$: a term t is typable in system $\mathcal{U}$ (or $\mathcal{U}$-typable) if and only if t terminates in vsc$_{\text{ne}}$. Later, in Section 5, this same type system $\mathcal{U}$ is shown to characterise termination of our notion of useful CBV evaluation (UCBV$^\bullet$) as well, thus establishing a relation between vsc$_{\text{ne}}$-termination, UCBV$^\bullet$-termination, and $\mathcal{U}$-typability.

3.1 Defining System $\mathcal{U}$

We begin with some preliminary considerations. First, normal forms in vsc$_{\text{ne}}$ and UCBV$^\bullet$ fall into three categories[2]. A normal form *without ESs* may be an *abstraction* $\lambda x.\, t$, a *variable* x, or a *structure* $x\, t_1 \dots t_n$, where $n \geq 1$ and each t_i is recursively in normal form. Normal forms *with ESs* also fit these categories up to performing all ESs. Thus, for instance, $(z\, y)[z\backslash x\, y]$ is a structure because it can be understood as representing the term $x\, y\, y$.

Second, one key feature of non-idempotent ITS is that a single occurrence of an expression in the (static) source code of a program may be (dynamically) used multiple times during execution, each time playing a different role, expressed by a different type. Specifically, each abstraction is assigned a multiset of arrow types of the form $[\alpha_1, \dots, \alpha_n]$, where the cardinality n corresponds to the number of times this abstraction[3] takes part as the left-hand side of a db-redex during evaluation. For example, in $(\lambda f.\, x\, (f\, y)\, (f\, z))\, \mathtt{I}$, the identity function takes part in *two* db-redexes during evaluation, so any type assignment for this term must provide *two* different arrow types for $\mathtt{I}$. In contrast, abstractions that are never applied, such as $\mathtt{I}$ in $x[x\backslash \mathtt{I}]$, are typed with the *empty* multiset $[\,]$.

As a general rule, terms whose normal form represents an abstraction – including variables bound to abstractions – are assigned finite multisets of arrow types, called **arrow multitypes**. On the other hand, terms whose normal form represents a structure – including applied variables and variables bound to structures – are assigned a distinguished type, $\mathbb{s}$. Formally, the types of system $\mathcal{U}$ are given by the following grammar:

---

[2] See Appendix A for the vsc$_{\text{ne}}$-normal forms and Section 4 for the UCBV$^\bullet$-normal forms.

[3] Or, more precisely, one of the *descendants* of this abstraction.

$$
\begin{array}{rlcll}
\textbf{(Arrow Types)} & \alpha, \beta & ::= & \mathcal{T}^? \to \mathcal{T} & \\
\textbf{(Arrow Multi-Types)} & \mathcal{M}, \mathcal{N}, \mathcal{M}_1, \mathcal{M}_2, \dots & ::= & [\alpha_i]_{i \in I} & \text{where } I \text{ is a finite set} \\
\textbf{(Types)} & \mathcal{T}, \mathcal{S}, \mathcal{T}_1, \mathcal{T}_2, \dots & ::= & \mathbb{s} \mid \mathcal{M} & \\
\textbf{(Optional Types)} & \mathcal{T}^?, \mathcal{S}^?, \mathcal{T}_1^?, \mathcal{T}_2^?, \dots & ::= & \bot \mid \mathcal{T} &
\end{array}
$$

A type $\mathcal{T}$ is said to be a **constant type** if $\mathcal{T} \in \{\mathbb{s}, []\}$, in which case we denote $\mathcal{T}$ by the letter $\mathbb{t}$. The type $\mathbb{s}$ is assigned to terms whose normal forms are structures, and $[]$ is assigned to terms whose normal forms are abstractions not taking part in db-redexes. Unlike in previous non-idempotent ITS for CBV, we distinguish the type $\bot$ from $[]$. The latter is used as explained above, while the former indicates that *no typing information is available.*

Typing environments $\Gamma, \Delta, \dots$ are functions mapping variables to optional types, assigning $\bot$ to all but finitely many variables. The **domain** of an environment Γ is $\mathsf{dom}(\Gamma) := \{x \mid \Gamma(x) \neq \bot\}$, and $\emptyset$ denotes the empty typing environment, mapping each variable to $\bot$.

The **union of arrow multi-types**, written $\mathcal{M}_1 \uplus \mathcal{M}_2$, is a multiset of types defined as expected, where $[]$ is the neutral element. The **union of types**, written $\mathcal{T}_1 + \mathcal{T}_2$, is the(associative) *partial* operation on types given by $\mathbb{s} + \mathbb{s} := \mathbb{s}$ and $\mathcal{M}_1 + \mathcal{M}_2 := \mathcal{M}_1 \uplus \mathcal{M}_2$ (where $\uplus$ denotes multiset union), with all other cases are undefined. The **union of optional types** is defined by $\bot + \bot := \bot$, $\bot + \mathcal{T} := \mathcal{T}$, and $\mathcal{T} + \bot := \mathcal{T}$, making $\bot$ the neutral element. For typing environments $(\Gamma_i)_{i \in I}$, we write $+_{i \in I} \Gamma_i$ for the environment mapping each variable x to $\uplus_{i \in I} \Gamma_i(x)$, where $\Gamma + \Delta$ and $\Gamma +_{j \in J} \Delta_j$ (meaning $\Gamma + (+_{j \in J} \Delta_j)$) are taken to be particular instances of the general notation. When $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset$ we write $\Gamma; \Delta$ instead of $\Gamma + \Delta$ to emphasise that the domains of Γ and Δ are disjoint. As a consequence, $\Gamma; x : \bot$ is identical to Γ.

Type assumptions are denoted $x : \mathcal{T}^?$, meaning that the environment assigns $\mathcal{T}^?$ to x, and $\bot$ to any other variable. We define a **controlled weakening** relation $\ll$ between optional types and types by declaring that $\bot \ll \mathbb{t}$ and $\mathcal{T} \ll \mathcal{T}$ hold.

Typing judgements in system $\mathcal{U}$ are of the form $\Gamma \vdash t : \mathcal{T}$, where Γ is a type environment and $\mathcal{T}$ is a type. **Typing rules** of system $\mathcal{U}$ are:

$$
\frac{}{x : \mathcal{T} \vdash x : \mathcal{T}}\,\textsc{var} \qquad \frac{\Gamma \vdash t : \mathbb{s} \quad \Delta \vdash u : \mathbb{t}}{\Gamma + \Delta \vdash t\, u : \mathbb{s}}\,\textsc{app}\mathrm{P} \qquad \frac{(\Gamma_i; x : \mathcal{T}_i^? \vdash t : \mathcal{S}_i)_{i \in I}}{+_{i \in I} \Gamma_i \vdash \lambda x.\, t : [\mathcal{T}_i^? \to \mathcal{S}_i]_{i \in I}}\,\textsc{abs}
$$

$$
\frac{\Gamma \vdash t : [\mathcal{T}^? \to \mathcal{S}] \quad \mathcal{T}^? \ll \mathcal{T} \quad \Delta \vdash u : \mathcal{T}}{\Gamma + \Delta \vdash t\, u : \mathcal{S}}\,\textsc{app}\mathrm{C} \qquad \frac{\Gamma; x : \mathcal{T}^? \vdash t : \mathcal{S} \quad \mathcal{T}^? \ll \mathcal{T} \quad \Delta \vdash u : \mathcal{T}}{\Gamma + \Delta \vdash t[x \backslash u] : \mathcal{S}}\,\textsc{es}
$$

A **(type) derivation** is a tree built by applying the previous (inductive) typing rules. We write $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathcal{T}$ to denote a type derivation Φ in system $\mathcal{U}$ ending in the typing judgement $\Gamma \vdash t : \mathcal{T}$. Sometimes, Φ or $\mathcal{U}$ may be omitted.

Like most non-idempotent ITS, system $\mathcal{U}$ is inspired by Linear Logic [31] and specifically by the relational semantics of CBV (see *e.g.*, [28]). Indeed, a multiset $[\alpha_1, \dots, \alpha_n]$ can be understood as a multiplicative conjunction $\alpha_1 \otimes \dots \otimes \alpha_n$. Moreover, there are no general *weakening* and *contraction* rules, though the facts that $\bot \ll \mathbb{t}$ and $\mathbb{t} + \mathbb{t} = \mathbb{t}$ hold whenever $\mathbb{t} \in \{\mathbb{s}, []\}$ can be understood respectively as controlled forms of weakening and contraction.

Rule ABS is a standard rule in non-idempotent ITS for CBV [28], and reflects the *weak* nature of reduction in $\mathrm{vsc}_{\mathrm{ne}}$ (reduction does not proceed below abstractions). For example, the derivation $\rhd \emptyset \vdash \lambda x.\, \Omega : []$ is valid even if Ω is non-terminating.

Rule ES can be easily reconstructed from rules ABS and APPC, while rules for applications follow the consuming/persistent paradigm [38, 39] and deserve some discussion. Indeed, system $\mathcal{U}$ keeps track of the different natures of the application constructors involved in the evaluation process of a program: a term constructor is **consuming** if it is *destroyed* during

evaluation, whereas a **persistent** constructor *remains* as a syntactical part of the normal form. For example, given the following reduction to normal form $x\,((\lambda y.\,y)\,z) \to_{\mathrm{vsc_{ne}}} x\,y[y\backslash z]$, we say that the leftmost application constructor in $x\,((\lambda y.\,y)\,z)$ is persistent, while the rightmost is consuming. Indeed, rule APPP (resp. APPC) is used to type this *persistent* (resp. *consuming*) application. More specifically, rule APPP requires the left-hand side of the application to have type $\mathbb{s}$, ensuring that no redex is created, so the application constructor is persistent. Variables that occur free at the top-level of a term are expected to be given the type $\mathbb{s}$, which means that the underlined applications in $\underline{z\,t}$ and $(\underline{x\,t})[x\backslash z]$ would type with rule APPP. Rule APPC requires the left-hand side of the application to type with an arrow multi-type, ensuring a db-redex is created at some point during evaluation. For instance, the underlined applications in $(\underline{x\,t})[x\backslash \mathtt{I}]$ and $(\underline{x\,t})[x\backslash y][y\backslash \mathtt{I}]$ would type with rule APPC.

As just explained, system $\mathcal{U}$ uses the consuming/persistent paradigm exclusively to type applications. This approach contrasts with more radical solutions [39] for CBV, which apply the same paradigm to type *all* constructors, thereby increasing the number of typing rules and making the type system more difficult to use.

The use of controlled weakening also deserves some discussion. Indeed, in CBV, the arguments of applications and ESs are always evaluated. Since system $\mathcal{U}$ intends to characterise termination, all arguments must necessarily be typed. For example, in $(\lambda x.\,\mathtt{I})\,(z\,z)$, the subexpression $z\,z$ is a structure of type $\mathbb{s}$. One would perhaps expect to be able to type the abstraction $\lambda x.\,\mathtt{I}$ with type $\mathbb{s} \to \mathcal{T}$. But it is impossible to derive the judgement $x : \mathbb{s} \vdash \mathtt{I} : \mathcal{T}$, because x does not occur free in $\mathtt{I}$. This is where the controlled weakening relation ($\ll$) in the premise of rules APPC and ES comes into play. In our example, one can give a "stronger" type to $\lambda x.\,\mathtt{I}$, namely $\bot \to \mathcal{T}$, using the fact that $\bot \ll \mathbb{s}$. A similar use of controlled weakening for the rule ES is shown below, where it should be recalled that the empty typing environment $\emptyset$ is identical to $\emptyset; x : \bot$.

$$\dfrac{\dfrac{}{\emptyset; x : \bot \vdash \mathtt{I} : [\,]}\,\textsc{abs} \qquad \bot \ll \mathbb{s} \qquad \dfrac{\dfrac{}{z : \mathbb{s} \vdash z : \mathbb{s}}\,\textsc{var} \qquad \dfrac{}{z : \mathbb{s} \vdash z : \mathbb{s}}\,\textsc{var}}{z : \mathbb{s} \vdash z\,z : \mathbb{s}}\,\textsc{appp}}{z : \mathbb{s} \vdash \mathtt{I}[x\backslash z\,z] : [\,]}\,\textsc{es}$$

We end this subsection with another example that will be revisited later in the paper.

▶ **Example 1.** If $t = (x\,z)[x\backslash y[y\backslash \mathtt{I}]]$ and $\mathcal{T} := [\mathbb{s} \to \mathbb{s}]$, we build the following derivation Φ:

$$\dfrac{\dfrac{\dfrac{}{x : \mathcal{T} \vdash x : \mathcal{T}}\,\textsc{var} \qquad \dfrac{}{z : \mathbb{s} \vdash z : \mathbb{s}}\,\textsc{var}}{z : \mathbb{s}, x : \mathcal{T} \vdash x\,z : \mathbb{s}}\,\textsc{appc} \qquad \dfrac{\dfrac{}{y : \mathcal{T} \vdash y : \mathcal{T}}\,\textsc{var} \qquad \dfrac{\dfrac{}{w : \mathbb{s} \vdash w : \mathbb{s}}\,\textsc{var}}{\emptyset \vdash \mathtt{I} : \mathcal{T}}\,\textsc{abs}}{\emptyset \vdash y[y\backslash \mathtt{I}] : \mathcal{T}}\,\textsc{es}}{z : \mathbb{s} \vdash (x\,z)[x\backslash y[y\backslash \mathtt{I}]] : \mathbb{s}}\,\textsc{es}$$

where $\mathcal{T} \ll \mathcal{T}$ and $\mathbb{s} \ll \mathbb{s}$ trivially hold.

3.2 Properties of System $\mathcal{U}$

The fact that the VAR typing rule in $\mathcal{U}$ is not weakened and that rules APPP, APPC, and ES are *multiplicative* (which is implemented by means of the operator $+$ on type environments) gives us a *relevance* property for system $\mathcal{U}$. Formally:

▶ **Lemma 2** (Relevance). *If* $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathcal{T}$, *then* $\mathsf{rv}(t) \subseteq \mathsf{dom}(\Gamma) \subseteq \mathsf{fv}(t)$.

Proof. See [16, Lemma 7.1]. ◀

Note that $\mathsf{fv}(t) \subseteq \mathsf{dom}(\Gamma)$ does not always hold. For instance, $\triangleright \emptyset \vdash \lambda x.\, y : [\,]$ but $\{y\} \not\subseteq \emptyset$.

Other specific properties of $\mathcal{U}$ are stated in Section 5. We end this section by stating the key property that the system $\mathcal{U}$ enjoys:

▶ **Theorem 3** (System $\mathcal{U}$ Characterises $\mathsf{vsc}_{\mathsf{ne}}$-Termination). *Let t be a term. Then, t is $\mathcal{U}$-typable if and only if t is $\mathrm{vsc}_{\mathrm{ne}}$-terminating.*

4 Useful Call-by-Value

This section recalls UCBV$^\bullet$, the inductive formulation of the useful open CBV strategy which provides a well-suited basis for semantic reasoning. Due to space limitations, we introduce only the main features of UCBV$^\bullet$; a detailed presentation can be found in [15, 16].

We begin by introducing the key notions of useful evaluation. Then we recall the operational semantics of UCBV$^\bullet$, and show some relevant properties of UCBV$^\bullet$, such as the inductive characterisation of UCBV$^\bullet$-normal forms (Theorem 5) and the diamond property (Theorem 6). The former ensures in particular that the length of any reduction sequence to normal form is independent of the specific order in which reduction rules are applied.

Useful Evaluation. The calculus with ESs used to implement sharing in Accattoli and Dal Lago's work [9] is based on a previous ES calculus with reduction at a distance called the *Linear Substitution Calculus* (LSC) [8]. In LSC, the reduction rule that implements function application is the same db-rule as in $\mathrm{vsc}_{\mathrm{ne}}$ (*cf.* subsection 2.2). Additionally, LSC uses a **linear substitution** rule, which only allows the substitution of a *single* occurrence of a variable x by a term t whenever x is bound to t by an ES. More precisely, the *linear substitution* rule (ls) of LSC allowing substitutions one occurrence at a time, is defined by the rewriting rule $\mathtt{C}\langle x\rangle[x\backslash t] \to_{\mathsf{ls}} \mathtt{C}\langle t\rangle[x\backslash t]$, where $\mathtt{C}$ denotes an arbitrary evaluation context. Substitution is called *linear* precisely because it replaces variables one occurrence at a time, for example $(x\,x)[x\backslash y] \to_{\mathsf{ls}} (x\,y)[x\backslash y] \to_{\mathsf{ls}} (y\,y)[x\backslash y]$.

Useful evaluation is based on two principles: *sharing structures* and (linearly) *substituting abstractions for progress*. We briefly recall them below.

Sharing Structures. In useful evaluation, a **structure** is a term whose *unfolding* – the result of performing all substitutions – results in an application headed by a variable. For example, $(x\,x)[x\backslash y\,\mathtt{I}][y\backslash z\,z]$ is a structure, as it unfolds to $z\,z\,\mathtt{I}\,(z\,z\,\mathtt{I})$. Structures must remain *shared* in useful evaluation, as substituting them does not create a db-redex. Thus, a term like $(x\,x)[x\backslash y\,\mathtt{I}][y\backslash z\,z]$ is a normal form in useful evaluation.

Substituting Abstractions for Progress. In useful evaluation, abstractions are substituted only if they contribute to creating a db-redex, ensuring progress in the computation. For example, the reduction step $x[x\backslash \mathtt{I}]\,y \to \mathtt{I}[x\backslash \mathtt{I}]\,y$ is useful because substituting x by $\mathtt{I}$ creates the db-redex $\mathtt{I}[x\backslash \mathtt{I}]\,y$. A subtler example of a useful step is $(x\,x)[x\backslash y][y\backslash \mathtt{I}] \to_{\mathsf{ls}} (y\,x)[x\backslash y][y\backslash \mathtt{I}]$, where the substitution *indirectly* contributes to creating a db-redex in the next step: $(y\,x)[x\backslash y][y\backslash \mathtt{I}] \to_{\mathsf{ls}} (\mathtt{I}\,x)[x\backslash y][y\backslash \mathtt{I}]$. Such indirect steps are precisely those that are challenging to capture inductively. On the other hand, steps like $x[x\backslash \mathtt{I}] \to \mathtt{I}[x\backslash \mathtt{I}]$, and $(x\,x)[x\backslash y\,y] \to_{\mathsf{ls}} ((y\,y)\,x)[x\backslash y\,y]$ and $(x\,x)[x\backslash \mathtt{I}] \to_{\mathsf{ls}} (x\,\mathtt{I})[x\backslash \mathtt{I}]$ are not useful, as these substitutions do not contribute to create a db-redex.

The principles just mentioned not only constitute the basis of useful CBN [9] but also that of useful CBV [1, 3], including UCBV$^\bullet$ [15].

Towards Useful Call-by-Value Evaluation. The key step to implement *(linear) substituting abstractions for progress* in UCBV$^\bullet$ is to define a family of reduction relations indexed by *parameters* capturing the essential information from the surrounding evaluation contexts:

this is the minimal data that cannot be ignored to decide which substitution step is useful and which is not. Two of these parameters are sets of variables: one is called an **(abstraction) frame** denoted $\mathcal{A}$ and the second is called a **(structure) frame** denoted $\mathcal{S}$. The purpose of these two kinds of variable sets is to classify the nature of variables bound by ESs.

Identifying terms that are already abstractions as well as those that are currently variables – but will ultimately be substituted with abstractions – is necessary to correctly determine whether a substitution step contributes to the progress of the evaluation. This property is captured by the following definition of **hereditary abstractions under the frame** $\mathcal{A}$, written $\mathsf{HA}_{\mathcal{A}}$:

$$\frac{x \in \mathcal{A}}{x \in \mathsf{HA}_{\mathcal{A}}} \qquad \frac{}{\lambda x.\, t \in \mathsf{HA}_{\mathcal{A}}} \qquad \frac{t \in \mathsf{HA}_{\mathcal{A}} \quad x \notin \mathcal{A}}{t[x\backslash u] \in \mathsf{HA}_{\mathcal{A}}} \qquad \frac{t \in \mathsf{HA}_{\mathcal{A}\cup\{x\}} \quad x \notin \mathcal{A} \quad u \in \mathsf{HA}_{\mathcal{A}}}{t[x\backslash u] \in \mathsf{HA}_{\mathcal{A}}}$$

For example, $\mathtt{I}[x\backslash y] \in \mathsf{HA}_{\emptyset}$ holds by the third rule, while $x[x\backslash \mathtt{I}] \in \mathsf{HA}_{\emptyset}$ holds by the fourth rule. Note that the fourth rule allows building "chains" such as $x[x\backslash y][y\backslash z[z\backslash \mathtt{I}]] \in \mathsf{HA}_{\emptyset}$.

It is also necessary to identify those terms that represent an application headed by a free variable, called the set of **structures under the frame** $\mathcal{S}$, written $\mathsf{St}_{\mathcal{S}}$, and defined as:

$$\frac{x \in \mathcal{S}}{x \in \mathsf{St}_{\mathcal{S}}} \qquad \frac{t \in \mathsf{St}_{\mathcal{S}} \quad x \notin \mathcal{S}}{t[x\backslash u] \in \mathsf{St}_{\mathcal{S}}} \qquad \frac{t \in \mathsf{St}_{\mathcal{S}}}{t\, u \in \mathsf{St}_{\mathcal{S}}} \qquad \frac{t \in \mathsf{St}_{\mathcal{S}\cup\{x\}} \quad x \notin \mathcal{S} \quad u \in \mathsf{St}_{\mathcal{S}}}{t[x\backslash u] \in \mathsf{St}_{\mathcal{S}}}$$

For example, $w[x\backslash y] \in \mathsf{St}_{\{w\}}$ holds by the second rule, $w\,(\mathtt{I}\,x) \in \mathsf{St}_{\{w\}}$ by the third rule, and $x[x\backslash w] \in \mathsf{St}_{\{w\}}$ by the fourth rule. Some properties of these predicates can be found in [15].

Useful Call-by-Value Reduction Rules. We can now recall the reduction rules for our useful CBV strategy, UCBV$^\bullet$. We first define a **step kind** as an element of the set $\{\mathsf{db}, \mathsf{lsv}, \mathsf{sub}_{(x,v)}\}$, with x being a variable and v a value such that $x \notin \mathsf{fv}(v)$. Let $\to_{\rho,\mathcal{A},\mathcal{S},\mu}$ be a family of binary relations, where ρ is a step kind, $\mathcal{A}$ is an abstraction frame, $\mathcal{S}$ a structure frame, and $\mu \in \{@, \not@\}$ is a constant called a **positional flag**. The positional flag @ means that the current focus of evaluation is in an applied position, while $\not@$ corresponds to a non-applied position. For example, the underlined x is applied in $\underline{x}[x\backslash \mathtt{I}]\,t$ but not in $(z\,\underline{x})[x\backslash \mathtt{I}]$. The relation $\to_{\rho,\mathcal{A},\mathcal{S},\mu}$[4] of UCBV$^\bullet$ is defined inductively as follows:

$$\frac{}{(\lambda x.\, t)\mathtt{L}\, u \to_{\mathsf{db},\mathcal{A},\mathcal{S},\mu} t[x\backslash u]\mathtt{L}}\,\mathrm{DB}^\bullet \qquad \frac{}{x \to_{\mathsf{sub}_{(x,v)},\mathcal{A}\cup\{x\},\mathcal{S},@} v}\,\mathrm{SUB}^\bullet$$

$$\frac{t \to_{\mathsf{sub}_{(x,v)},\mathcal{A}\cup\{x\},\mathcal{S},\mu} t' \quad x \notin \mathcal{A}\cup\mathcal{S} \quad v\mathtt{L} \in \mathsf{HA}_{\mathcal{A}}}{t[x\backslash v\mathtt{L}] \to_{\mathsf{lsv},\mathcal{A},\mathcal{S},\mu} t'[x\backslash v]\mathtt{L}}\,\mathrm{LSV}^\bullet$$

$$\frac{t \to_{\rho,\mathcal{A},\mathcal{S},@} t'}{t\, u \to_{\rho,\mathcal{A},\mathcal{S},\mu} t'\, u}\,\mathrm{APPL}^\bullet \qquad \frac{t \in \mathsf{St}_{\mathcal{S}} \quad u \to_{\rho,\mathcal{A},\mathcal{S},\not@} u'}{t\, u \to_{\rho,\mathcal{A},\mathcal{S},\mu} t\, u'}\,\mathrm{APPR}^\bullet \qquad \frac{u \to_{\rho,\mathcal{A},\mathcal{S},\not@} u'}{t[x\backslash u] \to_{\rho,\mathcal{A},\mathcal{S},\mu} t[x\backslash u']}\,\mathrm{ESR}^\bullet$$

$$\frac{t \to_{\rho,\mathcal{A}\cup\{x\},\mathcal{S},\mu} t' \quad u \in \mathsf{HA}_{\mathcal{A}} \quad x \notin \mathcal{A}\cup\mathcal{S} \quad x \notin \mathsf{fv}(\rho)}{t[x\backslash u] \to_{\rho,\mathcal{A},\mathcal{S},\mu} t'[x\backslash u]}\,\mathrm{ESLA}^\bullet$$

$$\frac{t \to_{\rho,\mathcal{A},\mathcal{S}\cup\{x\},\mu} t' \quad u \in \mathsf{St}_{\mathcal{S}} \quad x \notin \mathcal{A}\cup\mathcal{S} \quad x \notin \mathsf{fv}(\rho)}{t[x\backslash u] \to_{\rho,\mathcal{A},\mathcal{S},\mu} t'[x\backslash u]}\,\mathrm{ESLS}^\bullet$$

[4] In [15], this same relation is written $\xrightarrow{\bullet}_{\rho,\mathcal{A},\mathcal{S},\mu}$.

Reduction in UCBV$^\bullet$ is *weak*, *i.e.*, it is not allowed under abstractions, and it's *open*, since it allows reducing terms with free occurrences of variables. The specification of UCBV$^\bullet$ leaves the order of evaluation slightly underspecified. For example, in a term like $x\,t\,u$, the subterms t and u can be evaluated concurrently, without necessarily adhering to a strict (*e.g.* left-to-right) order. Technically speaking, the rules to evaluate applications are not mutually exclusive, and similarly for ESs. This non-determinism is harmless, as in fact UCBV$^\bullet$ enjoys the diamond property (*cf.* Theorem 6). For a concrete example in which rules APPL$^\bullet$ and APPR$^\bullet$ overlap, consider:

$$x\,(y[y\backslash \mathtt{I}])\,(\mathtt{I}\,\mathtt{I}) \leftarrow_{\mathsf{db},\emptyset,\{x\},\not\emptyset}\; x\,(\mathtt{I}\,\mathtt{I})\,(\mathtt{I}\,\mathtt{I}) \rightarrow_{\mathsf{db},\emptyset,\{x\},\not\emptyset}\; x\,(\mathtt{I}\,\mathtt{I})\,(y[y\backslash \mathtt{I}])$$

For a concrete example in which ESLS$^\bullet$ and ESR$^\bullet$ overlap, consider:

$$(y[y\backslash \mathtt{I}])[x\backslash z\,(\mathtt{I}\,\mathtt{I})] \leftarrow_{\mathsf{db},\emptyset,\{z\},\not\emptyset}\; (\mathtt{I}\,\mathtt{I})[x\backslash z\,(\mathtt{I}\,\mathtt{I})] \rightarrow_{\mathsf{db},\emptyset,\{z\},\not\emptyset}\; (\mathtt{I}\,\mathtt{I})[x\backslash z\,(y[y\backslash \mathtt{I}])]$$

As discussed in the introduction, useful evaluation is *context-sensitive*. That is, the context surrounding a term determines whether a step is useful or not. For instance, the term $t = x[x\backslash \mathtt{I}]$ is already a normal form for UCBV$^\bullet$ with respect to $\mathcal{A} = \emptyset$, $\mathcal{S} = \emptyset$, $\mu = \not\emptyset$ because substituting x by $\mathtt{I}$ is not useful, as it does not create a db-redex. However, if t appears on the left of an application, as in $x[x\backslash \mathtt{I}]\,(y\,\mathtt{I})$, the substitution of x by $\mathtt{I}$ becomes useful, leading to the following reduction sequence:

$$x[x\backslash \mathtt{I}]\,(y\,\mathtt{I}) \rightarrow_{\mathsf{lsv},\emptyset,\{y\},\not\emptyset}\; \mathtt{I}[x\backslash \mathtt{I}]\,(y\,\mathtt{I}) \rightarrow_{\mathsf{db},\emptyset,\{y\},\not\emptyset}\; z[z\backslash y\,\mathtt{I}][x\backslash \mathtt{I}]$$

Conversely, a useful step like $(x\,u)[x\backslash \mathtt{I}] \rightarrow_{\mathsf{lsv},\emptyset,\emptyset,\not\emptyset} (\mathtt{I}\,u)[x\backslash \mathtt{I}]$ may not be allowed if the term is located below a context such as $\lambda y.\,\Diamond$; in fact, note that $\lambda y.\,(x\,u)[x\backslash \mathtt{I}]$ cannot be reduced.

A term t is $(\rho, \mathcal{A}, \mathcal{S}, \mu)$**-irreducible** if there is no term t' such that $t \rightarrow_{\rho,\mathcal{A},\mathcal{S},\mu} t'$. A term t belongs to the set $\mathsf{Irred}^\bullet_{\mathcal{A},\mathcal{S},\mu}$ if t is $(\rho, \mathcal{A}, \mathcal{S}, \mu)$-irreducible.

An abstraction frame $\mathcal{A}$ and a structure frame $\mathcal{S}$ are said to verify the **correctness invariant** for t, written $\mathsf{inv}(\mathcal{A}, \mathcal{S}, t)$, if $\mathcal{A} \cap \mathcal{S} = \emptyset$ and $\mathsf{fv}(t) \subseteq \mathcal{A} \cup \mathcal{S}$. This invariant is typically assumed in theorems and lemmas. Note that $\mathsf{inv}(\emptyset, \mathsf{fv}(t), t)$ always holds, so for a **top-level** term t, we set $\mathcal{A} := \emptyset$ and $\mathcal{S} := \mathsf{fv}(t)$.

Rule SUB$^\bullet$ is an auxiliary tool to ensure that substitution in UCBV$^\bullet$ is *linear*. In general, we care about **top-level** UCBV$^\bullet$ **reduction**, defined as $\rightarrow_{\mathsf{top},\mathcal{S}} := \rightarrow_{\mathsf{db},\emptyset,\mathcal{S},\not\emptyset} \cup \rightarrow_{\mathsf{lsv},\emptyset,\mathcal{S},\not\emptyset}$.

▶ **Example 4.** The following is a UCBV$^\bullet$ reduction sequence to normal form:

$$\begin{aligned}(x\,z)[x\backslash y[y\backslash \mathtt{I}]] &\rightarrow_{\mathsf{lsv},\emptyset,\{z\},\not\emptyset}\; (y\,z)[x\backslash y][y\backslash \mathtt{I}] \\ \rightarrow_{\mathsf{lsv},\emptyset,\{z\},\not\emptyset}\; (\mathtt{I}\,z)[x\backslash y][y\backslash \mathtt{I}] &\rightarrow_{\mathsf{db},\emptyset,\{z\},\not\emptyset}\; x_1[x_1\backslash z][x\backslash y][y\backslash \mathtt{I}]\end{aligned}$$

To conclude this section, we recall some of the key operational properties of UCBV$^\bullet$, starting with a **characterisation of the set of ucbv$^\bullet$-normal forms** [15, Theorem 1]:

▶ **Theorem 5.** *The set of irreducible terms* $\mathsf{Irred}^\bullet_{\mathcal{A},\mathcal{S},\mu}$ *is exactly the set* $\mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\mu}$ *defined inductively as:*

$$\frac{x \in \mathcal{A} \text{ implies } \mu = \not\emptyset}{x \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\mu}} \qquad \frac{}{\lambda x.\,t \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\not\emptyset}} \qquad \frac{t \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},@} \quad u \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\not\emptyset}}{t\,u \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\mu}}$$

$$\frac{t \in \mathsf{NF}^\bullet_{\mathcal{A}\cup\{x\},\mathcal{S},\mu} \quad u \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\not\emptyset} \quad u \in \mathsf{HA}_{\mathcal{A}}}{t[x\backslash u] \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\mu}} \qquad \frac{t \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S}\cup\{x\},\mu} \quad u \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\not\emptyset} \quad u \in \mathsf{St}_{\mathcal{S}}}{t[x\backslash u] \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\mu}}$$

An important property of top-level UCBV$^\bullet$ is that it enjoys a strong form of confluence, the **Diamond Property** [15, Theorem 2]:

▶ **Theorem 6.** *Let* $t \to_{\rho_1,\emptyset,\mathcal{S},\not\oslash} t_1$ *and* $t \to_{\rho_2,\emptyset,\mathcal{S},\not\oslash} t_2$*, where* $t_1 \neq t_2$*, and* $\rho_1, \rho_2 \in \{\mathsf{db}, \mathsf{lsv}\}$*, and* $\mathcal{S} = \mathsf{fv}(t)$*. Then there exists* t' *such that* $t_1 \to_{\rho_2,\emptyset,\mathcal{S},\not\oslash} t'$ *and* $t_2 \to_{\rho_1,\emptyset,\mathcal{S},\not\oslash} t'$*.*

Some straightforward corollaries of the diamond property are, firstly, that a term t is weakly terminating in top-level UCBV$^\bullet$ if and only if t is terminating in top-level UCBV$^\bullet$. Secondly, that any two reduction sequences to normal form have the same number of db and lsv-steps. And finally, top-level UCBV$^\bullet$ reduction is immediately seen to be confluent. These corollaries are important for reaching our results in Section 5.

5 (Quantitative) Soundness and Completeness of System $\mathcal{U}$

In this section, we refine system $\mathcal{U}$ to capture quantitative information of UCBV$^\bullet$ evaluation. Indeed, we equip typing judgements of system $\mathcal{U}$ with counters, and we define a notion of tight derivation, intuitively corresponding to that of minimal derivation. The counters are meant to capture exact step-count for program evaluation. More precisely, tight derivations in system $\mathcal{U}$ not only guarantee termination in UCBV$^\bullet$ but also provide *exact* quantitative information about this evaluation strategy: we show that a term t is tightly typable with counters (m, e) if and only if t evaluates in UCBV$^\bullet$ to a normal form in exactly m function application steps (*i.e.*, db-steps) and e substitution steps (*i.e.*, lsv-steps). In this sense, we say that system $\mathcal{U}$ is a *quantitative interpretation* of useful CBV.

5.1 Tightness for System $\mathcal{U}$

As explained above, we start by refining the typing judgements of system $\mathcal{U}$ with counters. Formally, **typing judgements** are now of the form $\Gamma \vdash^{(m,e)} t : \mathcal{T}$, where m and e are natural numbers called **counters**. Under appropriate conditions, these counters correspond *exactly* to the number of db-steps (m) and lsv-steps (e) in UCBV$^\bullet$ evaluation, as shown in Theorems 10 and 13. Typing rules are refined accordingly. To do so, we introduce an operation $\mathtt{ta}(\_)$ that counts the number of top-level arrows in a type (or optional type). It is defined by $\mathtt{ta}(\bot) := 0$, $\mathtt{ta}(\mathtt{s}) := 0$, and $\mathtt{ta}([\alpha_i]_{i\in I}) := |I|$. Note that $\mathtt{ta}(\mathcal{T}_1 + \mathcal{T}_2) = \mathtt{ta}(\mathcal{T}_1) + \mathtt{ta}(\mathcal{T}_2)$. Formally, **typing rules** of system $\mathcal{U}$ are enriched with counters as follows:

$$\frac{n = \mathtt{ta}(\mathcal{T})}{x : \mathcal{T} \vdash^{(0,n)} x : \mathcal{T}}\text{VAR} \qquad \frac{\Gamma \vdash^{(m,e)} t : [\mathcal{T}^? \to \mathcal{S}] \quad \mathcal{T}^? \ll \mathcal{T} \quad \Delta \vdash^{(m',e')} u : \mathcal{T}}{\Gamma + \Delta \vdash^{(1+m+m',e+e')} t\,u : \mathcal{S}}\text{APPC}$$

$$\frac{\Gamma \vdash^{(m,e)} t : \mathtt{s} \quad \Delta \vdash^{(m',e')} u : \mathbb{t}}{\Gamma + \Delta \vdash^{(m+m',e+e')} t\,u : \mathtt{s}}\text{APPP} \qquad \frac{(\Gamma_i; x : \mathcal{T}_i^? \vdash^{(m_i,e_i)} t : \mathcal{S}_i)_{i\in I}}{+_{i\in I}\Gamma_i \vdash^{(+_{i\in I}m_i,+_{i\in I}e_i)} \lambda x.\,t : [\mathcal{T}_i^? \to \mathcal{S}_i]_{i\in I}}\text{ABS}$$

$$\frac{\Gamma; x : \mathcal{T}^? \vdash^{(m,e)} t : \mathcal{S} \quad \mathcal{T}^? \ll \mathcal{T} \quad \Delta \vdash^{(m',e')} u : \mathcal{T}}{\Gamma + \Delta \vdash^{(m+m',e+e')} t[x\backslash u] : \mathcal{S}}\text{ES}$$

Only two typing rules are increasing the counters. Rule VAR initialises the second counter to the number of top-level arrows in the type of the variable. This counter corresponds to the number of times the variable is (directly or indirectly) substituted by an abstraction taking part in some db-redex during evaluation. That is why the second counter tracks substitution step-count. Rule APPC increases the first counter by one, reflecting that the application constructor introduced by this rule is consuming, meaning it will be destroyed during evaluation. That is why the first counter tracks function application step-count.

▶ Remark 7. Counters are a convenient way to synthesise information *already existing* in a typing derivation: they do not impose any side conditions on derivations. Formally, $\rhd \Gamma \vdash t : \mathcal{T}$ if and only if $\rhd \Gamma \vdash^{(m,e)} t : \mathcal{T}$ for some (m, e). Moreover, the counters can be entirely extracted from the type information and typing rules of a derivation, no reference to the terms themselves is needed to compute them.

In what follows, we outline some basic properties of system $\mathcal{U}$.

Appropriateness. To reason inductively about typing derivations, it is sometimes necessary to ensure invariants for the types of the free variables in a term. For example, for a term like $t[x\backslash \mathtt{I}]$, we may need to track that x is bound to an abstraction when applying the inductive hypothesis for the subterm t. Specifically, we say that a typing environment Γ is **appropriate** with respect to an abstraction frame $\mathcal{A}$, written $\mathtt{appropriate}_{\mathcal{A}}(\Gamma)$ if, for each $x \in \mathcal{A}$, we have $\Gamma(x) \neq \mathbb{s}$. In other words, $\Gamma(x)$ must be $\bot$ or an arrow multi-type $\mathcal{M}$. For instance, $\mathtt{appropriate}_{\{x\}}(x : [\,], y : \mathbb{s})$ holds, while $\mathtt{appropriate}_{\{x\}}(x : \mathbb{s}, y : \mathbb{s})$ does not.

Tightness. Typing in system $\mathcal{U}$ not only characterises UCBV$^\bullet$ termination, but it is also capable of providing *exact measures* for reduction lengths to normal form. To state this formally, we define a subset of typing derivations, called **tight derivations**, as follows. A type is **tight** if it is a constant type $\mathbb{t}$, *i.e.*, either of the form $\mathbb{s}$ or $[\,]$. An optional type $\mathcal{T}^?$ is **tight** if it is either $\bot$ or a tight type. A typing environment Γ is **tight** if $\Gamma(x)$ is tight for every variable x. A typing judgement $\Gamma \vdash^{(m,e)} t : \mathcal{T}$ is **tight** if both Γ and $\mathcal{T}$ are tight. A derivation of a typing judgement is **tight** if the judgement itself is tight.

For example, the derivable judgement $x : [\,] \vdash^{(0,0)} x : [\,]$ is tight, where the counters $(0, 0)$ indicate that evaluating x requires no reduction steps, meaning x is already in normal form. In contrast, the judgement $x : [\mathcal{T} \to \mathcal{S}] \vdash^{(0,1)} x : [\mathcal{T} \to \mathcal{S}]$ is *not* tight, and the counters $(0, 1)$ provide an upper bound on the number of steps required for evaluating x.

▶ **Example 8.** Let $t = (x\, z)[x\backslash y[y\backslash \mathtt{I}]]$. Taking $\mathcal{T} := [\mathbb{s} \to \mathbb{s}]$, the following derivation Φ' (a decoration of Φ in Example 1) turns out to be tight:

$$\dfrac{\dfrac{\dfrac{}{x : \mathcal{T} \vdash^{(0,1)} x : \mathcal{T}}\,\text{VAR} \quad \dfrac{}{z : \mathbb{s} \vdash^{(0,0)} z : \mathbb{s}}\,\text{VAR}}{z : \mathbb{s}, x : \mathcal{T} \vdash^{(1,1)} x\, z : \mathbb{s}}\,\text{APP}\mathrm{C} \quad \dfrac{\dfrac{}{y : \mathcal{T} \vdash^{(0,1)} y : \mathcal{T}}\,\text{VAR} \quad \dfrac{\dfrac{}{w : \mathbb{s} \vdash^{(0,0)} w : \mathbb{s}}\,\text{VAR}}{\emptyset \vdash^{(0,0)} \mathtt{I} : \mathcal{T}}\,\text{ABS}}{\emptyset \vdash^{(0,1)} y[y\backslash \mathtt{I}] : \mathcal{T}}\,\text{ES}}{z : \mathbb{s} \vdash^{(1,2)} (x\, z)[x\backslash y[y\backslash \mathtt{I}]] : \mathbb{s}}\,\text{ES}$$

where $\mathbb{s} \ll \mathbb{s}$ and $\mathcal{T} = [\mathbb{s} \to \mathbb{s}] \ll [\mathbb{s} \to \mathbb{s}] = \mathcal{T}$ both hold.

Note that the only subderivations of Φ' which are tight (besides Φ' itself) correspond to the one concluding in $z : \mathbb{s} \vdash^{(0,0)} z : \mathbb{s}$ and the one concluding in $w : \mathbb{s} \vdash^{(0,0)} w : \mathbb{s}$. The remaining subderivations are not tight.

Tightness is essential for proving that system $\mathcal{U}$ is sound and complete with respect to UCBV$^\bullet$ (Theorems 10 and 13).

5.2 Soundness and Completeness

We first address the **soundness** of system $\mathcal{U}$ with respect to UCBV$^\bullet$: typability implies UCBV$^\bullet$-termination. This proof follows well-understood techniques, requiring a subject reduction property based on a substitution lemma. Due to the *contextual* nature of UCBV$^\bullet$, these lemmas are not trivial, as they demand formulating complex invariants involving the contextual parameters $\mathcal{A}$, $\mathcal{S}$, and μ that define the evaluation strategy.

We first show subject reduction, stating that each UCBV$^\bullet$ reduction step preserves typing and correctly decrements the counters.

▶ **Proposition 9** (Subject Reduction). *Let* $t \to_{\rho,\mathcal{A},\mathcal{S},\mu} t'$ *where* $\rho \in \{\mathsf{db}, \mathsf{lsv}\}$ *and* $\Phi \rhd \Gamma \vdash^{(m,e)} t : \mathcal{T}$, *with* $\mathtt{appropriate}_{\mathcal{A}}(\Gamma)$. *Suppose moreover that if* $\mu = @$, *then either* $\mathcal{T} = \mathbb{s}$ *or* $\mathcal{T}$ *is a singleton,* i.e., *of the form* $[\alpha]$. *Then, there exists a derivation* Φ' *such that* $\Phi' \rhd \Gamma \vdash^{(m',e')} t' : \mathcal{T}$, *where if* $\rho = \mathsf{db}$, *then* $m > 0$ *and* $(m', e') = (m-1, e)$, *and if* $\rho = \mathsf{lsv}$, *then* $e > 0$ *and* $(m', e') = (m, e-1)$.

Proof. See [16, Proposition 7.3]. ◀

We can now conclude with a quantitative version of soundness: a tight typing derivation of a term t with counters (m, e) guarantees a terminating UCBV$^\bullet$ reduction sequence, with exactly m db-steps and e lsv-steps.

▶ **Theorem 10** (Quantitative Soundness of System $\mathcal{U}$). *Let* $\mathcal{S} = \mathsf{fv}(t)$, *and let* $\Phi \rhd \Gamma \vdash^{(m,e)} t : \mathcal{T}$. *Then, there exists a* $\to_{\mathtt{top},\mathcal{S}}$*-irreducible term* u *such that* t *reduces to* u. *Moreover, if* Φ *is tight, then* $t \to^{m+e}_{\mathtt{top},\mathcal{S}} u$, *where* m *and* e *are respectively the number of* db *and* lsv *steps in the previous reduction sequence.*

To illustrate this property, consider the tight derivation Φ for $t = (x\,z)[x\backslash y[y\backslash \mathtt{I}]]$ in Example 8, where the counter in the conclusion of Φ is $(1, 2)$. A reduction sequence from t terminates in $x_1[x_1\backslash z][x\backslash y][y\backslash \mathtt{I}] \in \mathsf{NF}^\bullet_{\emptyset,\{z\},@}$ in exactly 1 db-steps and 2 lsv-steps (see Example 4). Note how these numbers match the previous counters.

Tightness in the previous example is crucial. Indeed, non-tight judgements may not provide exact information about the length of reduction sequences to normal form. For example, the derivation of the judgement $z : \mathbb{s}, x : \mathcal{T} \vdash^{(1,1)} x\,z : \mathbb{s}$ in Example 8 is not tight, as $\mathcal{T}$ is not tight, and the counters $(1, 1)$ do not correspond to the number of steps needed to reach the normal form of $x\,z$ which is already in normal form.

Completeness of system $\mathcal{U}$ can be proved using standard techniques, requiring a subject expansion property. These statements are quite technical, as the properties must be generalised for inductive reasoning.

The first property guarantees that normal forms are tightly typable in system $\mathcal{U}$. For that, we need to construct tight environments for each normal form $t \in \mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\mu}$; this is done by typing the *reachable* variables in $\mathcal{A}$ with $[\,]$ and those in $\mathcal{S}$ with $\mathbb{s}$. More precisely, given t, $\mathcal{A}$, and $\mathcal{S}$ such that $\mathsf{inv}(\mathcal{A}, \mathcal{S}, t)$, the **tight environment** for t under $\mathcal{A}$ and $\mathcal{S}$ is written $\mathsf{TEnv}(\mathcal{A}, \mathcal{S}, t)$ and defined as the environment Γ such that $\Gamma(x) = [\,]$ when $x \in \mathcal{A} \cap \mathsf{rv}(t)$, $\Gamma(x) = \mathbb{s}$ when $x \in \mathcal{S} \cap \mathsf{rv}(t)$, and $\Gamma(x) = \bot$ otherwise. Tight environments allow us to type normal forms *tightly*.

▶ **Proposition 11** (Normal Forms are Tight Typable). *Let* t *be a term in* $\mathsf{NF}^\bullet_{\mathcal{A},\mathcal{S},\mu}$ *and* $\mathsf{inv}(\mathcal{A}, \mathcal{S}, t)$. *Then, there exists a tight type* $\mathbb{t}$ *such that* $\rhd \mathsf{TEnv}(\mathcal{A}, \mathcal{S}, t) \vdash^{(0,0)} t : \mathbb{t}$. *Moreover, if* $t \in \mathsf{HA}_{\mathcal{A}}$, *then* $\mathbb{t} = [\,]$, *and if* $t \in \mathsf{St}_{\mathcal{S}}$, *then* $\mathbb{t} = \mathbb{s}$.

Proof. See [16, Proposition 7.5]. ◀

Subject Expansion is analogous to Subject Reduction but applies in the *reverse direction*: given a typable term t' and a reduction step $t \to_{\rho,\mathcal{A},\mathcal{S},\mu} t'$, the term t is typable as well, with the same type and typing environment.

▶ **Proposition 12** (Subject Expansion). *Let* $\mathsf{inv}(\mathcal{A}, \mathcal{S}, t)$ *hold, and suppose* $t \to_{\rho,\mathcal{A},\mathcal{S},\mu} t'$ *where* $\rho \in \{\mathsf{db}, \mathsf{lsv}\}$ *and* $\Phi' \rhd \Gamma \vdash^{(m',e')} t' : \mathcal{T}$*, with* $\mathtt{appropriate}_{\mathcal{A}}(\Gamma)$*. Furthermore, assume that if* $\mu = @$*, then either* $\mathcal{T} = \mathbb{s}$ *or* $\mathcal{T}$ *is a singleton,* i.e.*, of the form* $[\alpha]$*. Then, there exists a derivation* Φ *such that* $\Phi \rhd \Gamma \vdash^{(m,e)} t : \mathcal{T}$*, where if* $\rho = \mathsf{db}$*, then* $(m, e) = (m' + 1, e')$*, and if* $\rho = \mathsf{lsv}$*, then* $(m, e) = (m', e' + 1)$*.*

Proof. See [16, Proposition 7.6]. ◀

We now conclude with a quantitative version of completeness of system $\mathcal{U}$. That is, given a terminating UCBV$^\bullet$ reduction sequence from t with exactly m db-steps and e lsv-steps, there exists a tight derivation of t with counters (m, e):

▶ **Theorem 13** (Quantitative Completeness of System $\mathcal{U}$). *Let* $\mathcal{S} = \mathsf{fv}(t)$ *and consider a reduction sequence* $t \to^n_{\mathtt{top},\mathcal{S}} u$*, where* u *is* $\to_{\mathtt{top},\mathcal{S}}$*-irreducible. Let* $n = m + e$*, where* m *and* e *are respectively the numbers of* db *and* lsv *steps in the sequence. Then, there exists a tight environment* Γ *and a tight type* $\mathbb{t}$ *such that* $\rhd \Gamma \vdash^{(m,e)} t : \mathbb{t}$*.*

To illustrate this result, take the reduction sequence starting from $t = (x\, z)[x\backslash y[y\backslash \mathtt{I}]]$ and ending in $x_1[x_1\backslash z][x\backslash y][y\backslash \mathtt{I}] \in \mathsf{NF}^\bullet_{\emptyset,\{z\},@}$ (see Example 4). The length of this sequence is $3 = m + e$, with $m = 1$ db-step and $e = 2$ lsv-steps. Example 8 shows a tight environment $z : \mathbb{s}$ and a tight type $\mathbb{s}$ such that $\Phi \rhd z : \mathbb{s} \vdash^{(1,2)} t : \mathbb{s}$.

Theorems 10 and 13 state that a term t is typable in system $\mathcal{U}$ if and only if t is *weakly terminating*. The characterisation of UCBV$^\bullet$ termination through typability follows from these theorems and the fact that all reduction sequences to normal form have the same length in UCBV$^\bullet$ (an easy corollary of Theorem 6). Moreover, this characterisation is quantitative:

▶ **Corollary 14** (Quantitative Characterisation of Termination). *A term* t *is tightly typable with counters* (m, e) *in system* $\mathcal{U}$ *if and only if* t *terminates in* UCBV$^\bullet$ *after* m *function application* db*-steps and* e *substitution* lsv*-steps.*

We can now conclude with the main contribution of this work.

▶ **Corollary 15.** *Let* t *be a term. The following statements are equivalent:*

1. t *is typable in system* $\mathcal{U}$*.*
2. t *terminates in* vsc*.*
3. t *terminates in* $\mathrm{vsc}_{\mathrm{ne}}$*.*
4. t *terminates in top-level* UCBV$^\bullet$ *reduction.*

Proof. (**1** $\Leftrightarrow$ **3**) is Theorem 3. (**1** $\Rightarrow$ **4**) Any $\mathcal{U}$-typable term terminates in UCBV$^\bullet$ by Proposition 9. (**4** $\Rightarrow$ **1**) Immediate by Theorem 13. Therefore (**1** $\Leftrightarrow$ **3** $\Leftrightarrow$ **4**).

Subject reduction for vsc *fails* with respect to $\mathcal{U}$. Hence, to prove (**3** $\Leftrightarrow$ **2**) we resort to an alternative type system $\mathcal{V}$ [19], inspired by Ehrhard's system [28]. Indeed, a term t is typable in system $\mathcal{V}$ if and only if t terminates in vsc [19, Theorem 5]. It can also be shown that system $\mathcal{V}$ characterises termination in $\mathrm{vsc}_{\mathrm{ne}}$ (see Appendix B for details), thus concluding the last equivalence[5]. ◀

We can conclude this section by emphasising an important consequence. As usual, a **context** $\mathtt{C}$ is a term with a single occurrence of a *hole* $\Diamond$, and $\mathtt{C}\langle t\rangle$ denotes the term obtained by replacing the hole in $\mathtt{C}$ by t. Given a reduction system $\mathcal{R}$, we define t to

[5] An interesting additional consequence is that systems $\mathcal{U}$ and $\mathcal{V}$ type the same terms.

be **observationally equivalent** to u, written $t \cong_{\mathcal{R}} u$, if and only if for every context $\mathtt{C}$ we have that $\mathtt{C}\langle t\rangle$ $\mathcal{R}$-terminates if and only if $\mathtt{C}\langle u\rangle$ $\mathcal{R}$-terminates. Then $\mathcal{R}_1$ and $\mathcal{R}_2$ are **observationally equivalent** if and only if for every t, u, we have $t \cong_{\mathcal{R}_1} u \iff t \cong_{\mathcal{R}_2} u$.

Corollary 15 above establishes in particular that the (potentially) erasing vsc, and the non-erasing vsc and top-level UCBV$^\bullet$ strategy are pairwise observational equivalent.

6 Conclusions

This paper presents the first semantic model of useful evaluation in the literature, based on the non-idempotent intersection type system $\mathcal{U}$. Even though the specification of the operational semantics of useful CBV evaluation is highly complex, system $\mathcal{U}$ is notably simple. This highlights the ability of semantic approaches based on non-idempotent intersection types to capture intricate operational details with streamlined intuitive methods.

System $\mathcal{U}$ characterises termination of the UCBV$^\bullet$ evaluation strategy as well as termination of vsc_{ne}. This ensures in particular that useful CBV evaluation can be seen as an optimisation of CBV that preserves its original semantics: they are observationally equivalent.

Moreover, system $\mathcal{U}$ provides exact measures for the lengths of reduction sequences in UCBV$^\bullet$ to normal form; that is why system $\mathcal{U}$ is a *quantitative* interpretation of UCBV$^\bullet$. More precisely, a term t is *tightly* typable with counters m and e in system $\mathcal{U}$ if and only if t terminates in UCBV$^\bullet$ in exactly m function application steps and e substitution steps.

Our contribution is novel, as previous definitions of useful evaluation (including useful CBV) in the literature lack semantic models, and current quantitative interpretations of CBV do not consider usefulness. Moreover, with the exception of [39], entirely based on the (verbose) persistent/consuming paradigm, this is the only *simple* and *concise* quantitative type system for open CBV that is able to count all kind of substitution steps (variables and abstractions) exactly.

Several lines of work stem from this paper. One first objective is to understand the flexibility of system $\mathcal{U}$, particularly its potential to capture some *other optimisations* of CBV, such as those in [5]. Additionally, it would be interesting to capture recent alternative specifications of usefulness, such as those in [45], which focus exclusively on *direct* useful reduction steps. Notably, erasing reduction steps in [45] pose a non-trivial difficulty for non-idempotent type systems designed for usefulness. Another challenging direction is to extend our inductive formalisation of useful evaluation to *strong* CBV, thus allowing reductions to take place inside abstractions. The methodology we have used to design system $\mathcal{U}$ could then be applied to such extensions.

References

1 Beniamino Accattoli and Claudio Sacerdoti Coen. On the relative usefulness of fireballs. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.23.

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

3 Beniamino Accattoli and Giulio Guerrieri. Implementing open call-by-value. In Mehdi Dastani and Marjan Sirjani, editors, *Fundamentals of Software Engineering - 7th International Conference, FSEN 2017, Tehran, Iran, April 26-28, 2017, Revised Selected Papers*, volume 10522 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2017. doi:10.1007/978-3-319-68972-2_1.

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

5 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019. doi:10.1016/J.SCICO.2019.03.002.

6 Beniamino Accattoli and Giulio Guerrieri. The theory of call-by-value solvability. *Proc. ACM Program. Lang.*, 6(ICFP):855–885, 2022. doi:10.1145/3547652.

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

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

9 Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 8:1–8:10. ACM, 2014. doi:10.1145/2603088.2603105.

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

11 Sandra Alves, Delia Kesner, and Miguel Ramos. Quantitative global memory. In Helle Hvid Hansen and Andre Scedrov, editors, *Logic, Language, Information, and Computation - 29th International Workshop, WoLLIC 2023, Halifax, Canada, September 20-23, 2023, Proceedings*, Lecture Notes in Computer Science, 2023.

12 Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In Ron K. Cytron and Peter Lee, editors, *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 233–246. ACM Press, 1995. doi:10.1145/199448.199507.

13 Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. Foundations of strong call by need. *Proc. ACM Program. Lang.*, 1(ICFP):20:1–20:29, 2017. doi:10.1145/3110264.

14 Pablo Barenbaum, Eduardo Bonelli, and Kareem Mohamed. Pattern matching and fixed points: Resource types and strong call-by-need: Extended abstract. In David Sabel and Peter Thiemann, editors, *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*, pages 6:1–6:12. ACM, 2018. doi:10.1145/3236950.3236972.

15 Pablo Barenbaum, Delia Kesner, and Mariana Milicich. A fresh inductive approach to useful call-by-value. In Clark W. Barrett and Uwe Waldmann, editors, *Automated Deduction - CADE 30 - 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings*, volume 15943 of *Lecture Notes in Computer Science*, pages 381–399. Springer, 2025. doi:10.1007/978-3-031-99984-0_21.

16 Pablo Barenbaum, Delia Kesner, and Mariana Milicich. Useful evaluation: Syntax and semantics (technical report), 2025. arXiv:2404.18874.

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

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

19 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. In Keisuke Nakano and Konstantinos Sagonas, editors, *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*, volume 12073 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2020. doi:10.1007/978-3-030-59025-3_2.

20 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. *Inf. Comput.*, 293:105047, 2023. doi:10.1016/J.IC.2023.105047.

21 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Log. J. IGPL*, 25(4):431–464, 2017. doi:10.1093/JIGPAL/JZX018.

22 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul.* PhD thesis, Ecole Doctorale Physique et Sciences de la Matière (Marseille), 2007.

23 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978. doi:10.1007/BF02011875.

24 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Log.*, 21(4):685–693, 1980. doi:10.1305/NDJFL/1093883253.

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

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

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

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

29 Álvaro García-Pérez and Pablo Nogueira. No solvable lambda-value term left behind. *Log. Methods Comput. Sci.*, 12(2), 2016. doi:10.2168/LMCS-12(2:12)2016.

30 Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software*, pages 555–574. Springer, 1994. doi:10.1007/3-540-57887-0_115.

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

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

33 Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In Mitchell Wand and Simon L. Peyton Jones, editors, *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002*, pages 235–246. ACM, 2002. doi:10.1145/581478.581501.

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

35 Delia Kesner. A theory of explicit substitutions with safe and full composition. *Logical Methods in Computer Science*, 5(3), 2009. URL: http://arxiv.org/abs/0905.2539.

36 Delia Kesner. Reasoning about call-by-need by means of types. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 2016. doi:10.1007/978-3-662-49630-5_25.

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

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

39 Delia Kesner and Andrés Viso. Encoding tight typing in a unified framework. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.27.

40 Maico Leberle. *Dissecting call-by-need by customizing multi type systems. (Une dissection de l'appel-par-nécessité par la personnalisation des systèmes de multi types)*. PhD thesis, IP Paris, France, 2021. URL: https://tel.archives-ouvertes.fr/tel-03284370.

41 John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. In Stephen D. Brookes, Michael G. Main, Austin Melton, and Michael W. Mislove, editors, *Eleventh Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 1995, Tulane University, New Orleans, LA, USA, March 29 - April 1, 1995*, volume 1 of *Electronic Notes in Theoretical Computer Science*, pages 370–392. Elsevier, 1995. doi:10.1016/S1571-0661(04)00022-2.

42 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO Theor. Informatics Appl.*, 33(6):507–534, 1999. doi:10.1051/ITA:1999130.

43 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

44 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

45 Jui-Hsuan Wu. Proofs as terms, terms as graphs. In Chung-Kil Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 91–111. Springer, 2023. doi:10.1007/978-981-99-8311-7_5.

A Proofs of Section 3

Measures. Given an optional type $\mathcal{T}^?$, we define its **size** as $|\bot| := 0$, $|\mathbb{s}| := 0$ and $|[\alpha_k]_{k\in K}| := |K|$. We remark that $|\mathcal{T}_1^? + \mathcal{T}_2^?| = |\mathcal{T}_1^?| + |\mathcal{T}_2^?|$.

In this subsection, we measure typing derivations in system $\mathcal{U}$ by using a function $\mathtt{sz}\,(\_)$, defined inductively as follows:

$$\mathtt{sz}\left(\frac{}{x:\mathcal{T}\vdash x:\mathcal{T}}\textsc{var}\right) := |\mathcal{T}| \qquad \mathtt{sz}\left(\frac{(\Phi_i)_{i\in I}}{\Gamma\vdash \lambda x.\,u : [\alpha_i]_{i\in I}}\textsc{abs}\right) := |I| + \sum_{i\in I}\mathtt{sz}\,(\Phi_i)$$

$$\mathtt{sz}\left(\frac{\Phi_1 \qquad \Phi_2}{\Gamma\vdash u\,s:\mathcal{T}}\textsc{appP}/\textsc{appC}\right) := 1 + \mathtt{sz}\,(\Phi_1) + \mathtt{sz}\,(\Phi_2)$$

$$\mathtt{sz}\left(\frac{\Phi_1 \qquad \Phi_2}{\Gamma\vdash u[x\backslash s]:\mathcal{T}}\textsc{es}\right) := \begin{cases} 1 + \mathtt{sz}\,(\Phi_1) + \mathtt{sz}\,(\Phi_2) & \text{if } x\in \mathsf{fv}(u) \\ \mathtt{sz}\,(\Phi_1) + \mathtt{sz}\,(\Phi_2) & \text{otherwise} \end{cases}$$

When a typing derivation Φ ends with rule ES, $\mathtt{sz}(\Phi)$ is defined according to two different cases. Whenever $x \in \mathsf{fv}(u)$, then a substitution of x by u takes place, hence we add 1 to the sum of the sizes of the derivations of both t and u.

We first address the **soundness** of system $\mathcal{U}$ with respect to $\mathrm{vsc}_{\mathrm{ne}}$, which states that $\mathcal{U}$-typability implies $\mathrm{vsc}_{\mathrm{ne}}$-termination. This proof follows well-understood techniques, requiring a subject reduction property based on a substitution lemma. For proving the latter, we need an additional lemma allowing the decomposition of the multiset types of values:

▶ **Lemma 16** (Splitting/Merging). *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two types such that their union $\mathcal{T}_1 + \mathcal{T}_2$ is well-defined. Then, the following are equivalent:*

1. $\Phi \rhd \Gamma \vdash v : \mathcal{T}_1 + \mathcal{T}_2$
2. *There exist typing derivations Φ_1, Φ_2 and typing contexts Γ_1, Γ_2 such that $\Phi_1 \rhd \Gamma_1 \vdash v : \mathcal{T}_1$, and $\Phi_2 \rhd \Gamma_2 \vdash v : \mathcal{T}_2$, and $\Gamma = \Gamma_1 + \Gamma_2$.*

Moreover, $\mathtt{sz}(\Phi) = \mathtt{sz}(\Phi_1) + \mathtt{sz}(\Phi_2)$.

Proof. The proof corresponds to that of [16, Lemma E.3], and it is straightforward to prove that $\mathtt{sz}(\Phi) = \mathtt{sz}(\Phi_1) + \mathtt{sz}(\Phi_2)$. ◀

▶ **Lemma 17** (Type Splitting). *Let $\mathcal{S}_1^?$, $\mathcal{S}_2^?$ be two optional types and $\mathcal{S}$ be a type. Then $\mathcal{S}_1^? + \mathcal{S}_2^?$ is well-defined and $\mathcal{S}_1^? + \mathcal{S}_2^? \ll \mathcal{S}$ if and only if there exist types $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 + \mathcal{S}_2 = \mathcal{S}$ and $\mathcal{S}_i^? \ll \mathcal{S}_i$ for all $i \in \{1, 2\}$.*

Proof. Both directions of the lemma follow by case analysis on $\mathcal{S}_1^?$ and $\mathcal{S}_2^?$. ◀

▶ **Lemma 18** (Substitution). *Let $\Phi_t \rhd_{\mathcal{U}} \Gamma; x : \mathcal{S}^? \vdash t : \mathcal{T}$ and $\Phi_v \rhd_{\mathcal{U}} \Delta \vdash v : \mathcal{S}$, with $\mathcal{S}^? \ll \mathcal{S}$ and $x \in \mathsf{fv}(t)$. Then, there exists a typing derivation $\Phi_{t\{x:=v\}}$ such that $\Phi_{t\{x:=v\}} \rhd_{\mathcal{U}} \Gamma + \Delta \vdash t\{x := v\} : \mathcal{T}$ and moreover $\mathtt{sz}\left(\Phi_{t\{x:=v\}}\right) = \mathtt{sz}(\Phi_t) + \mathtt{sz}(\Phi_v) - |\mathcal{S}^?|$.*

Proof. By induction on Φ_t. We show only case VAR, as the inductive cases follow easily from the *i.h.* and Lemmas 16 and 17.

If Φ_t ends with VAR, then $t = x$, since $x \in \mathsf{fv}(t)$ by hypothesis. Then, $t\{x := v\} = v$ and Φ_t is of the form $x : \mathcal{T} \vdash x : \mathcal{T}$, thus $\Gamma = \emptyset$ and $\mathcal{S}^? = \mathcal{S} = \mathcal{T}$. We let $\Phi_{t\{x:=v\}} := \Phi_v$ and we are done. Moreover, $\mathtt{sz}\left(\Phi_{t\{x:=v\}}\right) = \mathtt{sz}(\Phi_v) = |\mathcal{T}| + \mathtt{sz}(\Phi_v) - |\mathcal{T}| = \mathtt{sz}(\Phi_t) + \mathtt{sz}(\Phi_v) - |\mathcal{S}^?|$. ◀

The condition $x \notin \mathsf{fv}(t)$ in the lemma above is necessary, since the statement would not hold otherwise. As a counterexample, take $v = z$, $\Delta = z : \mathbb{t}$, and $t = y$ with $y \neq x$.

▶ Remark 19. If $\Phi \rhd_{\mathcal{U}} \Gamma \vdash v : \mathbb{t}$, then $\mathtt{sz}(\Phi) = 0$.

▶ **Lemma 20** (Weighted Subject Reduction). *Let $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathcal{T}$ and $t \to_{\mathrm{vsc}_{\mathrm{ne}}} t'$. Then, there exists a derivation Φ' such that $\Phi' \rhd_{\mathcal{U}} \Gamma \vdash t' : \mathcal{T}$ and moreover $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.*

Proof. By induction on $t \to_{\mathrm{vsc}_{\mathrm{ne}}} t'$. We only show the base case $t \to_{\mathsf{svne}} t'$. Let $t = u[x\backslash v\mathtt{L}] \to_{\mathsf{svne}} u\{x := v\}[x\backslash v]\mathtt{L} = t'$, with $x \in \mathsf{fv}(u)$. We proceed by induction on $\mathtt{L}$.

- $\mathtt{L} = \Diamond$. Then, Φ has the form:

$$\frac{\Phi_u \rhd \Gamma_1; x : \mathcal{S}^? \vdash u : \mathcal{T} \qquad \mathcal{S}^? \ll \mathcal{S} \qquad \Phi_v \rhd \Gamma_2 \vdash v : \mathcal{S}}{\Gamma_1 + \Gamma_2 \vdash u[x\backslash v] : \mathcal{T}}\,\mathrm{ES}$$

  where $\Gamma = \Gamma_1 + \Gamma_2$. Notice that $\mathcal{S}^? = \mathcal{S}^? + \bot$. Thus, let $\mathcal{S}_1^? = \mathcal{S}^?$ and $\mathcal{S}_2^? = \bot$. We separate in cases depending on whether $\mathcal{S} = \mathbb{s}$ or $\mathcal{S} = \mathcal{M}$, showing only the former. Hence $\mathcal{S} = \mathbb{s}$, and by Lemma 17 there exist $\mathcal{S}_1 = \mathbb{s}$ and $\mathcal{S}_2 = \mathbb{s}$ such that $\mathcal{S} = \mathbb{s} = \mathbb{s} + \mathbb{s} = \mathcal{S}_1 + \mathcal{S}_2$ (by idempotency of the union of constant types), and $\mathcal{S}^? \ll \mathbb{s}$ and $\bot \ll \mathbb{s}$. Recall that $|\mathbb{s}| = |\bot| = 0$ by definition, so in particular $|\mathcal{S}^?| = |\mathcal{S}_1^?| = 0$.

Applying Lemma 16, we obtain $\Phi_v^1 \rhd \Gamma_{21} \vdash v : \mathbb{s}$ and $\Phi_v^2 \rhd \Gamma_{22} \vdash v : \mathbb{s}$ with $\Gamma_2 = \Gamma_{21} + \Gamma_{22}$ and $\mathtt{sz}\left(\Phi_v^1\right) + \mathtt{sz}\left(\Phi_v^2\right) = \mathtt{sz}\left(\Phi_v\right) = 0$ by Remark 19. We can apply Lemma 18 on Φ_u and Φ_v^1, yielding $\Phi_{u\{x:=v\}} \rhd \Gamma_1 + \Gamma_{21} \vdash u\{x := v\} : \mathcal{T}$ and $\mathtt{sz}\left(\Phi_{u\{x:=v\}}\right) = \mathtt{sz}\left(\Phi_u\right) + \mathtt{sz}\left(\Phi_v^1\right) - |\mathcal{S}_1^?| = \mathtt{sz}\left(\Phi_u\right)$. Thus, we conclude building Φ':

$$\dfrac{\Phi_{u\{x:=v\}} \rhd (\Gamma_1 + \Gamma_{21}); x : \bot \vdash u\{x := v\} : \mathcal{T} \quad \bot \ll \mathbb{s} \quad \Phi_v^2 \rhd \Gamma_{22} \vdash v : \mathbb{s}}{\Gamma_1 + \Gamma_2 \vdash u\{x := v\}[x\backslash v] : \mathcal{T}}\text{ES}$$

And $\mathtt{sz}\left(\Phi\right) = 1 + \mathtt{sz}\left(\Phi_u\right) + \mathtt{sz}\left(\Phi_v\right) = 1 + \mathtt{sz}\left(\Phi_u\right) > \mathtt{sz}\left(\Phi_u\right) = \mathtt{sz}\left(\Phi_{u\{x:=v\}}\right) = \mathtt{sz}\left(\Phi_{u\{x:=v\}}\right) + \mathtt{sz}\left(\Phi_v^2\right) = \mathtt{sz}\left(\Phi'\right)$.

- $\mathtt{L} = \mathtt{L}'[y\backslash s]$. Then, Φ has the form:

$$\dfrac{\Phi_u \rhd \Gamma'; x : \mathcal{S}^? \vdash u : \mathcal{T} \quad \mathcal{S}^? \ll \mathcal{S} \qquad \dfrac{\Phi_{v\mathtt{L}'} \rhd \Delta_1; y : \mathcal{R}^? \vdash v\mathtt{L}' : \mathcal{S} \quad \mathcal{R}^? \ll \mathcal{R} \quad \Phi_s \rhd \Delta_2 \vdash s : \mathcal{R}}{\Delta_1 + \Delta_2 \vdash v\mathtt{L}'[y\backslash s] : \mathcal{S}}\text{ES}}{\Gamma' + \Delta_1 + \Delta_2 \vdash u[x\backslash v\mathtt{L}'[y\backslash s]] : \mathcal{T}}\text{ES}$$

where $\Gamma = \Gamma' + \Delta_1 + \Delta_2$. Moreover, we can assume $y \notin \mathsf{fv}(u)$ by α-conversion.
We now build the following derivation Ψ:

$$\dfrac{\Phi_u \rhd \Gamma'; x : \mathcal{S}^? \vdash u : \mathcal{T} \quad \mathcal{S}^? \ll \mathcal{S} \quad \Phi_{v\mathtt{L}'} \rhd \Delta_1; y : \mathcal{R}^? \vdash v\mathtt{L}' : \mathcal{S}}{\Gamma' + (\Delta_1; y : \mathcal{R}^?) \vdash u[x\backslash v\mathtt{L}'] : \mathcal{T}}\text{ES}$$

Moreover, $u[x\backslash v\mathtt{L}'] \to_{\mathsf{svne}} u\{x := v\}[x\backslash v]\mathtt{L}'$, since $x \in \mathsf{fv}(u)$. Then, we can apply the *i.h.*, yielding Ψ' such that $\Psi' \rhd \Gamma' + (\Delta_1; y : \mathcal{R}^?) \vdash u\{x := v\}[x\backslash v]\mathtt{L}' : \mathcal{T}$ and moreover, $\mathtt{sz}\left(\Psi\right) > \mathtt{sz}\left(\Psi'\right)$. Then, we build Φ' as follows:

$$\dfrac{\Psi' \rhd \Gamma' + (\Delta_1; y : \mathcal{R}^?) \vdash u\{x := v\}[x\backslash v]\mathtt{L}' : \mathcal{T} \quad \mathcal{R}^? \ll \mathcal{R} \quad \Phi_s \rhd \Delta_2 \vdash s : \mathcal{R}}{\Gamma' + \Delta_1 + \Delta_2 \vdash u\{x := v\}[x\backslash v]\mathtt{L}'[y\backslash s] : \mathcal{T}}\text{ES}$$

Note that we can assume $y \notin \mathsf{dom}(\Gamma')$ by α-conversion and Relevance.
To compare the sizes of both Φ and Φ', we consider two cases, depending on whether $y \in \mathsf{fv}(u\{x := v\}[x\backslash v]\mathtt{L}')$ or not. Note that $x \in \mathsf{fv}(u)$ by hypothesis.

- If $y \notin \mathsf{fv}(u\{x := v\}[x\backslash v]\mathtt{L}')$, then in particular $y \notin \mathsf{fv}(v\mathtt{L}')$. Thus, we have $\mathtt{sz}\left(\Phi\right) = 1 + \mathtt{sz}\left(\Phi_u\right) + \mathtt{sz}\left(\Phi_{v\mathtt{L}'}\right) + \mathtt{sz}\left(\Phi_s\right) = \mathtt{sz}\left(\Psi\right) + \mathtt{sz}\left(\Phi_s\right) >_{i.h.} \mathtt{sz}\left(\Psi'\right) + \mathtt{sz}\left(\Phi_s\right) = \mathtt{sz}\left(\Phi'\right)$.
- Otherwise, $y \in \mathsf{fv}(u\{x := v\}[x\backslash v]\mathtt{L}')$, and recall that $y \notin \mathsf{fv}(u)$ by α-conversion, so that $y \in \mathsf{fv}(v\mathtt{L}')$. Therefore, $\mathtt{sz}\left(\Phi\right) = 1 + \mathtt{sz}\left(\Phi_u\right) + 1 + \mathtt{sz}\left(\Phi_{v\mathtt{L}'}\right) + \mathtt{sz}\left(\Phi_s\right) = 1 + \mathtt{sz}\left(\Psi\right) + \mathtt{sz}\left(\Phi_s\right) >_{i.h.} 1 + \mathtt{sz}\left(\Psi'\right) + \mathtt{sz}\left(\Phi_s\right) = \mathtt{sz}\left(\Phi'\right)$. ◀

We now prove the **completeness** of system $\mathcal{U}$ with respect to $\mathrm{vsc}_{\mathrm{ne}}$, which states that $\mathrm{vsc}_{\mathrm{ne}}$-termination implies $\mathcal{U}$-typability. This result follows well-understood techniques, requiring a subject expansion property based on an anti-substitution lemma.

▶ **Lemma 21** (Anti-Substitution). *Let $\Phi_{t\{x:=v\}} \rhd_{\mathcal{U}} \Gamma \vdash t\{x := v\} : \mathcal{T}$, with $x \in \mathsf{fv}(t)$. Then, there exist typing derivations Φ_t and Φ_v, typing contexts Γ_t and Δ, an optional type $\mathcal{S}^?$, and a type $\mathcal{S}$ satisfying:*

1. $\Phi_t \rhd_{\mathcal{U}} \Gamma_t; x : \mathcal{S}^? \vdash t : \mathcal{T}$
2. $\Phi_v \rhd_{\mathcal{U}} \Delta \vdash v : \mathcal{S}$
3. $\Gamma = \Gamma_t + \Delta$ *and* $\mathcal{S}^? \ll \mathcal{S}$

Moreover, $\mathtt{sz}\left(\Phi_{t\{x:=v\}}\right) = \mathtt{sz}\left(\Phi_t\right) + \mathtt{sz}\left(\Phi_v\right) - |\mathcal{S}^?|$.

Proof. By structural induction on t. We only show the case when t is a variable. The inductive cases follow easily from the *i.h.* and Lemma 16.

When t is a variable, then necessarily $t = x$, as $x \in \mathsf{fv}(t)$ by hypothesis. Then, $t\{x := v\} = v$. We let $\Phi_v := \Phi_{t\{x:=v\}}$, and we yield $\Phi_t \rhd x : \mathcal{T} \vdash x : \mathcal{T}$ by rule VAR, so that $\Gamma_t := \emptyset$, $\Delta := \Gamma$, and $\mathcal{S}^? = \mathcal{S} = \mathcal{T}$. Moreover, $\mathtt{sz}\left(\Phi_{t\{x:=v\}}\right) = \mathtt{sz}\left(\Phi_v\right) = |\mathcal{S}| + \mathtt{sz}\left(\Phi_v\right) - |\mathcal{S}| = \mathtt{sz}\left(\Phi_t\right) + \mathtt{sz}\left(\Phi_v\right) - |\mathcal{S}^?|$. ◀

The condition $x \notin \mathsf{fv}(t)$ from the previous lemma is necessary, since otherwise, the statement does not hold. As a counterexample, it is sufficient to take $t = y$ with $y \neq x$ and $v = z$ so that $t\{x := z\} = y$ and the derivation $\Phi_{t\{x:=z\}} \rhd y : \mathcal{T} \vdash y : \mathcal{T}$.

▶ **Lemma 22** (Weighted Subject Expansion). *Let $t \to_{\mathrm{vsc_{ne}}} t'$ and $\Phi' \rhd_{\mathcal{U}} \Gamma \vdash t' : \mathcal{T}$. Then, there exists Φ such that $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathcal{T}$, with $\mathtt{sz}\left(\Phi\right) > \mathtt{sz}\left(\Phi'\right)$.*

Proof. By induction on $t \to_{\mathrm{vsc_{ne}}} t'$ where, in particular, Lemma 21 is used in the base case $t = u[x\backslash v\mathtt{L}] \to_{\mathsf{sv}} u\{x := v\}[x\backslash v]\mathtt{L} = t'$. ◀

To prove completeness of system $\mathcal{U}$ with respect to $\mathrm{vsc_{ne}}$ evaluation, we first establish a lemma showing that normal forms are $\mathcal{U}$-typable. We start by defining a grammar of terms and then showing that this grammar characterises the set of $\mathrm{vsc_{ne}}$-normal forms.

$$\begin{array}{lcl} \mathtt{vr} & ::= & x \mid \mathtt{vr}[x\backslash\mathtt{ne}] \mid \mathtt{vr}[x\backslash\mathtt{no}] \ (x \notin \mathsf{fv}(\mathtt{vr})) \\ \mathtt{ne} & ::= & \mathtt{vr}\,\mathtt{no} \mid \mathtt{ne}\,\mathtt{no} \mid \mathtt{ne}[x\backslash\mathtt{ne}] \mid \mathtt{ne}[x\backslash\mathtt{no}] \ (x \notin \mathsf{fv}(\mathtt{ne})) \\ \mathtt{ab} & ::= & \lambda x.\, t \mid \mathtt{ab}[x\backslash\mathtt{ne}] \mid \mathtt{ab}[x\backslash\mathtt{no}] \ (x \notin \mathsf{fv}(\mathtt{ab})) \\ \mathtt{no} & ::= & \mathtt{vr} \mid \mathtt{ne} \mid \mathtt{ab} \end{array}$$

We define three predicates $\mathsf{abs}(\_)$, $\mathsf{app}(\_)$, and $\mathsf{var}(\_)$ we will use for the characterisation of $\mathrm{vsc_{ne}}$-normal forms. For any term t, the predicate $\mathsf{abs}(t)$ holds if and only $t = (\lambda x.\, u)\mathtt{L}$ for some variable x, some term u and some substitution context $\mathtt{L}$; the predicate $\mathsf{app}(t)$ holds if and only $t = (u\, s)\mathtt{L}$ for some terms u, s and some substitution context $\mathtt{L}$; and the predicate $\mathsf{var}(t)$ holds if and only $t = x\mathtt{L}$ for some variable x and some substitution context $\mathtt{L}$.

▶ **Proposition 23** (Characterisation of $\mathrm{vsc_{ne}}$-Normal Forms). *Let t be a term. Then, $t \in \mathtt{no}$ if and only if t is $\mathrm{vsc_{ne}}$-irreducible.*

Proof. The proof is by simultaneous induction on the following statements:

1. $t \in \mathtt{vr}$ if and only if t is $\mathrm{vsc_{ne}}$-irreducible and $\mathsf{var}(t)$
2. $t \in \mathtt{ne}$ if and only if t is $\mathrm{vsc_{ne}}$-irreducible and $\mathsf{app}(t)$
3. $t \in \mathtt{ab}$ if and only if t is $\mathrm{vsc_{ne}}$-irreducible and $\mathsf{abs}(t)$
4. $t \in \mathtt{no}$ if and only if t is $\mathrm{vsc_{ne}}$-irreducible ◀

▶ **Lemma 24** (Normal Forms in $\mathrm{vsc_{ne}}$ are $\mathcal{U}$-Typable). *Let t be a $\mathrm{vsc_{ne}}$-normal form. Then, t is $\mathcal{U}$-typable with a constant type $\mathbb{t}$. More specifically, there exist Φ, Γ, and $\mathbb{t}$ such that $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathbb{t}$, where $\Gamma(x) = \mathbb{s}$ for all $x \in \mathsf{dom}(\Gamma)$.*

Proof. The proof is by simultaneous induction on the following statements:

1. If $t \in \mathtt{vr}$, then there are Φ and Γ such that $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathbb{s}$, and $\Gamma(y) = \mathbb{s}$ for all $y \in \mathsf{dom}(\Gamma)$.
2. If $t \in \mathtt{ne}$, then there are Φ and Γ such that $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathbb{s}$, and $\Gamma(y) = \mathbb{s}$ for all $y \in \mathsf{dom}(\Gamma)$.
3. If $t \in \mathtt{ab}$, then there are Φ and Γ such that $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : [\,]$, and $\Gamma(y) = \mathbb{s}$ for all $y \in \mathsf{dom}(\Gamma)$.
4. If $t \in \mathtt{no}$, then there are Φ, Γ, and $\mathbb{t}$ such that $\Phi \rhd_{\mathcal{U}} \Gamma \vdash t : \mathbb{t}$, and $\Gamma(y) = \mathbb{s}$ for all $y \in \mathsf{dom}(\Gamma)$. ◀

We now state the result of the characterisation of $\mathrm{vsc_{ne}}$-termination via $\mathcal{U}$-typability:

▶ **Theorem 3** (System $\mathcal{U}$ Characterises $\mathrm{vsc}_{\mathrm{ne}}$-Termination). *Let t be a term. Then, t is $\mathcal{U}$-typable if and only if t is $\mathrm{vsc}_{\mathrm{ne}}$-terminating.*

Proof. The *only if* direction holds by Lemma 20, while the *if* direction follows from Lemmas 22 and 24. ◀

B Proofs of Section 5

In this subsection, we detail the equivalence **3** $\Leftrightarrow$ **2** in the proof of Corollary 15.

The Type System $\mathcal{V}$. Consider a set of base types α. The grammar of types in system $\mathcal{V}$ is given by:

$$\begin{array}{rrcll} \textbf{(Types)} & \sigma,\tau & ::= & \alpha \mid \mathcal{M} \mid \mathcal{M} \to \sigma & \\ \textbf{(Multi-Types)} & \mathcal{M},\mathcal{N} & ::= & [\sigma_i]_{i\in I} & \text{where } I \text{ is a finite set} \end{array}$$

The typing rules of system $\mathcal{V}$ are:

$$\frac{}{x : \mathcal{M} \vdash x : \mathcal{M}}\,\mathrm{VAR_v} \qquad \frac{(\Gamma_i; x : \mathcal{M}_i \vdash t : \sigma_i)_{i\in I}}{+_{i\in I}\Gamma_i \vdash \lambda x.\, t : [\mathcal{M}_i \to \sigma_i]_{i\in I}}\,\mathrm{ABS_v}$$

$$\frac{\Gamma \vdash t : [\mathcal{M} \to \sigma] \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t\, u : \sigma}\,\mathrm{APP_v} \qquad \frac{\Gamma; x : \mathcal{M} \vdash t : \sigma \quad \Delta \vdash u : \mathcal{M}}{\Gamma + \Delta \vdash t[x\backslash u] : \sigma}\,\mathrm{ES_v}$$

A term t is **$\mathcal{V}$-typable** if there is a derivation in $\mathcal{V}$ for typing t, written $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : \sigma$.

We measure typing derivations in system $\mathcal{V}$ using the $\mathtt{sz}(\_)$ function, which is defined following the same intuitions we used for the homonym function in Appendix A. Specifically:

$$\mathtt{sz}\left(\frac{}{x : \mathcal{T} \vdash x : \mathcal{T}}\,\mathrm{VAR_v}\right) := |\mathcal{T}| \qquad \mathtt{sz}\left(\frac{(\Phi_i)_{i\in I}}{\Gamma \vdash \lambda x.\, t : [\alpha_i]_{i\in I}}\,\mathrm{ABS_v}\right) := |I| + \sum_{i\in I} \mathtt{sz}(\Phi_i)$$

$$\mathtt{sz}\left(\frac{\Phi_1 \quad \Phi_2}{\Gamma \vdash t\, u : \mathcal{T}}\,\mathrm{APP_v}\right) := 1 + \mathtt{sz}(\Phi_1) + \mathtt{sz}(\Phi_2)$$

$$\mathtt{sz}\left(\frac{\Phi_1 \quad \Phi_2}{\Gamma \vdash t[x\backslash u] : \mathcal{T}}\,\mathrm{ES_v}\right) := \begin{cases} 1 + \mathtt{sz}(\Phi_1) + \mathtt{sz}(\Phi_2) & \text{if } x \in \mathsf{fv}(u) \\ \mathtt{sz}(\Phi_1) + \mathtt{sz}(\Phi_2) & \text{otherwise} \end{cases}$$

We start addressing the **soundness** of system $\mathcal{V}$ with respect to $\mathrm{vsc}_{\mathrm{ne}}$, which states that $\mathcal{V}$-typability implies $\mathrm{vsc}_{\mathrm{ne}}$-termination. As with the results for system $\mathcal{U}$ in Appendix A, the soundness result requires a subject reduction property based on a substitution lemma.

▶ Remark 25. For every value v there exists a derivation $\Phi \rhd_{\mathcal{V}} \emptyset \vdash v : [\,]$ such that $\mathtt{sz}(\Phi) = 0$.

▶ **Lemma 26** (Substitution). *Let $\Phi_t \rhd_{\mathcal{V}} \Gamma; x : \mathcal{M} \vdash t : \sigma$ and $\Phi_v \rhd_{\mathcal{V}} \Delta \vdash v : \mathcal{M}$. Then, there exists a typing derivation $\Phi_{t\{x:=v\}}$ such that $\Phi_{t\{x:=v\}} \rhd_{\mathcal{V}} \Gamma + \Delta \vdash t\{x := v\} : \sigma$ and moreover $\mathtt{sz}\left(\Phi_{t\{x:=v\}}\right) = \mathtt{sz}(\Phi_t) + \mathtt{sz}(\Phi_v) - |\mathcal{M}|$.*

Proof. See [20, Lemma 4.18]. ◀

▶ **Lemma 27** (Weighted Subject Reduction). *Let $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : \sigma$ and $t \to_{\mathrm{vsc}_{\mathrm{ne}}} t'$. Then, there exists a derivation Φ' such that $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash t' : \sigma$ and moreover $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.*

Proof. Using Lemma 26, the proof is analogous to the proof of Lemma 20 in Appendix A. ◀

We now prove the **completeness** of system $\mathcal{V}$ with respect to $\mathrm{vsc}_{\mathrm{ne}}$, which states that $\mathrm{vsc}_{\mathrm{ne}}$-termination implies $\mathcal{V}$-typability. As with the results for system $\mathcal{U}$ in Appendix A, we require to state a subject expansion property based on an anti-substitution lemma.

▶ **Lemma 28** (Anti-Substitution). *Let $\Phi_{t\{x:=v\}} \rhd_{\mathcal{V}} \Gamma \vdash t\{x := v\} : \sigma$. Then, there exist typing derivations Φ_t and Φ_v, typing contexts Γ_t and Δ, and a type $\mathcal{S}$ satisfying:*
1. $\Phi_t \rhd_{\mathcal{V}} \Gamma_t; x : \mathcal{M} \vdash t : \sigma$
2. $\Phi_v \rhd_{\mathcal{V}} \Delta \vdash v : \mathcal{M}$
3. $\Gamma = \Gamma_t + \Delta$

Moreover, $\mathtt{sz}\left(\Phi_{t\{x:=v\}}\right) = \mathtt{sz}\left(\Phi_t\right) + \mathtt{sz}\left(\Phi_v\right) - |\mathcal{M}|$.

Proof. See [20, Lemma 4.18]. ◀

▶ **Lemma 29** (Weighted Subject Expansion). *Let $t \to_{\mathrm{vsc}_{\mathrm{ne}}} t'$ and $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash t' : \mathcal{T}$. Then, there exists Φ such that $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : \mathcal{T}$, with $\mathtt{sz}\left(\Phi\right) > \mathtt{sz}\left(\Phi'\right)$.*

Proof. By induction on $t \to_{\mathrm{vsc}_{\mathrm{ne}}} t'$ where, in particular, Lemma 28 is used in the base case $t = u[x\backslash v\mathtt{L}] \to_{\mathsf{sv}} u\{x := v\}[x\backslash v]\mathtt{L} = t'$. ◀

▶ **Lemma 30** (Normal Forms in $\mathsf{vsc}_{\mathsf{ne}}$ are $\mathcal{V}$-Typable). *Let t be a $\mathrm{vsc}_{\mathrm{ne}}$-normal form. Then, t is $\mathcal{V}$-typable.*

Proof. The proof proceeds by simultaneous induction on the following statements:
1. If $t \in \mathtt{vr}$, then for every multi-type $\mathcal{M}$ there exist Φ and Γ such that $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : \mathcal{M}$.
2. If $t \in \mathtt{ne}$, then for every type σ there exist Φ and Γ such that $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : \sigma$.
3. If $t \in \mathtt{ab}$, then there exist Φ and Γ such that $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : [\,]$.
4. If $t \in \mathtt{no}$, then there exist Φ, Γ, and $\mathcal{M}$ such that $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t : \mathcal{M}$. ◀

We now state the result of the characterisation of $\mathrm{vsc}_{\mathrm{ne}}$-termination via $\mathcal{V}$-typability:

▶ **Theorem 31.** *Let t be a term. Then, t is $\mathrm{vsc}_{\mathrm{ne}}$-terminating if and only if t is $\mathcal{V}$-typable.*

Proof. The *only if* (resp. *if*) direction follows from Lemma 27 (resp. from Lemmas 29 and 30). ◀