Reasoning about call-by-need by means of types

Delia Kesner

Univ. Paris-Diderot, SPC, IRIF, CNRS, France

Abstract. We first develop a (semantical) characterization of call-by-need normalization by means of typability, *i.e.* we show that a term is normalizing in call-by-need if and only if it is typable in a suitable system with non-idempotent intersection types. This first result is used to derive a new completeness proof of call-by-need w.r.t. call-by-name. Concretely, we show that call-by-need and call-by-name are observationally equivalent, so that in particular, the former turns out to be a correct implementation of the latter.

1 Introduction

There is a real gap between the well-known operational semantics of the *call-by-name* λ-calculus and the actual implementations of *lazy* functional languages such as Miranda or Haskell. Indeed, call-by-name re-evaluates an argument each time it is used – an operation which is quite expensive – while lazy languages store the *value* of an argument the first time it is evaluated, thus avoiding the need for any subsequent re-evaluations. For example, consider the term $t_0 = \Delta(II)$, where $\Delta = \lambda x.xx$ and $I = \lambda z.z$. Call-by-name duplicates the argument II, while lazy languages first reduces II to the value I so that further uses of this argument do not need any more to evaluate it again.

However, lazy languages should not be confused with *call-by-value*: values are only consumed in the former when they are *needed/required*, thus completely ignoring the other arguments. *E.g.* if $t_1 = (\lambda z.I)(II)$, the argument II is evaluated in call-by-value to produce the final answer I, whereas this evaluation does not take place in any lazy language (see *e.g.* [4, 5, 13, 17]). Said differently, lazy languages do not compute unneeded terms, and terms should not be duplicated unless and until they have been reduced to values.

According to the previous paragraphs, lazy languages cannot be modeled by call-by-name, nor by call-by-value; they are indeed specified by *call-by-need*, due to Wadsworth [39], and extensively studied *e.g.* in [2, 4, 5, 13, 17, 32].

Despite the fact that the equational theories of call-by-name and call-by-need are not the same [4], call-by-need is observationally equivalent (*i.e.* sound and complete w.r.t.) to call-by-name. The completeness result, operationally developed *e.g.* in [4, 32], is quite involved and makes use of different *syntactical* tools such as commutation diagrams, sharing, residual theory and standardization.

In this paper we make use of *typing systems* to provide a *semantical* argument showing that call-by-need and call-by-name are observationally equivalent. More precisely, we develop a new completeness proof of call-by-need w.r.t. call-by-name

which follows from two different implications: every term having a normal form in call-by-name is typable and every typable term has a normal form in call-by-need. The second implication is obtained from a stronger characterization property stating that the set of typable terms and the set of call-by-need normalizing terms are the same. This result is completely new, and quite surprising, and constitutes one of the main contributions of this work.

In order to achieve our goal we use intersection types, originally introduced in [15], being able to provide models of the λ-calculus [7], and characterizations of head [14] as well as weakly [12, 14] and strongly normalizing λ-terms [28]. Typically, intersection types are *idempotent* [7]. However, in the last years, growing interest has been devoted to *non-idempotent* intersection types [9, 25], since they allow for reasoning about *quantitative* properties of terms, both from a syntactical and a semantical point of view. Different assignment systems with non-idempotent intersection types have been studied in the literature for different purposes [8, 10, 18, 19, 24, 26, 27, 30, 35, 34].

Reasoning about call-by-name and call-by-need is achieved in this paper by introducing two (non-idempotent) typing systems, called $\mathcal{V}$ and $\mathcal{A}$ respectively. System $\mathcal{V}$ (resp. $\mathcal{A}$) characterizes the set of `name` (resp. `need`) normalizing terms, *i.e.* a term t is typable in system $\mathcal{V}$ (resp. $\mathcal{A}$) if and only if t is normalizing in call-by-name (resp. call-by-need). While our characterization of call-by-name normalization can be seen as a simple adaptation of the existing one for head-normalization [19], this is, to the best of our knowledge, the first time that call-by-need normalization is characterized by means of semantical/logical tools.

Non-idempotent intersection types provide a full characterizations of call-by-name/need normalizing terms by using very simple combinatorial arguments, no reducibility technique are used. More precisely, there is a measure, defined on type derivations, which *strictly* decreases during name/needed reduction. This gives quantitative information about call-by-name/need reduction sequences.

The following list summarizes the more important points of our work:

1. We provide a characterization of `name`-normalization by adapting an existing one for head-normalization.
2. We develop a completely new characterization of `need`-normalization by means of typability. This characterization, as well as the one for `name`-normalization, is arithmetical, *i.e.* no reducibility arguments are needed.
3. Based on the previous points, we show that call-by-need and call-by-name are observationally equivalent, so that in particular, we obtain a new (semantical) completeness proof of call-by-need w.r.t. call-by-name.

Related works: Besides all the aforementioned works that use intersection types to characterize operational properties of λ-calculi, two other works make use of typing systems to reason about lazy languages. In [31], the *resource conscious* character of linear logic is used to explain call-by-need in the spirit of the Curry-Howard isomorphism. In [20] non-idempotent intersection types are related to *needed redexes*: a redex r in a term t is *needed* iff r, or some *residual* of r, is reduced in *every* reduction of t to normal form. Despite the use of

similar typing techniques to identify needed redexes in well-typed terms, no full characterization of needed normalization is obtained in [20].

Structure of the paper: Sec. 2 introduces syntax and operational semantics of call-by-name and call-by-need. Sec. 3 defines non-idempotent type assignment disciplines for both calculi. Sec. 4 (resp. Sec. 5) gives a characterization of `name` (resp. `need`)-normalizing terms by means of the intersection type systems introduced in Sec. 3. Sec. 6 presents the completeness proof by using the semantical/logical arguments developed in the previous sections. Sec. 7 concludes and suggests some further directions of research.

2 Call-by-name and call-by-need

This section describes the syntax and the operational semantics of our languages. We start by introducing call-by-name in the setting of the lambda-calculus, as done in the seminal work of Plotkin [36]. Given a countable infinite set of symbols $x, y, z, \ldots$, we define terms, values and name contexts as follows:

Terms	$t, u ::= x \mid tu \mid \lambda x.t$
Values	$v ::= \lambda x.t$
Name contexts	$\mathtt{E} ::= \Box \mid \mathtt{E}\,t$

The set of terms is denoted by $\mathcal{T}_{\mathtt{a}}$. We write $\mathtt{I}$ for the identity function $\lambda x.x$. The sets of **free** and **bound** variables of a term t, written respectively $\mathtt{fv}(t)$ and $\mathtt{bv}(t)$, are defined as usual. We work with the standard notion of **α-conversion** *i.e.* renaming of bound variables for abstractions. **Substitutions** are (finite) functions from variables to terms denoted by $\{x_1/u_1, \ldots, x_n/u_n\}$ ($n \geq 0$). **Application** of the **substitution** σ to the term t, written $t\sigma$, is defined on α-equivalence classes, as usual, so that capture of free variables cannot hold.

The **call-by-name calculus** is given by the set of terms $\mathcal{T}_{\mathtt{a}}$ and the **reduction relation** $\to_{\mathtt{name}}$, which is the closure by *name* contexts $\mathtt{E}$ of the rewriting rule $(\lambda x.t)u \mapsto_\beta t\{x/u\}$. An example of `name`-reduction sequence is

$$\begin{array}{l}(\lambda f.f\mathtt{I}(f\mathtt{I}))(\lambda w.(\mathtt{II})w) \to_{\mathtt{name}} (\lambda w.(\mathtt{II})w)\mathtt{I}((\lambda w.(\mathtt{II})w)\mathtt{I}) \to_{\mathtt{name}} \\ (\mathtt{II})\mathtt{I}((\lambda w.(\mathtt{II})w)\mathtt{I}) \quad \twoheadrightarrow^3_{\mathtt{name}} (\lambda w.(\mathtt{II})w)\mathtt{I} \to_{\mathtt{name}} (\mathtt{II})\mathtt{I} \to_{\mathtt{name}} \mathtt{II} \to_{\mathtt{name}} \mathtt{I}\end{array}$$

Remark that $t \to_{\mathtt{name}} t'$ implies $\mathtt{fv}(t) \supseteq \mathtt{fv}(t')$, and notice that reduction is *weak*, so that we never reduce under λ-abstractions, which are considered to be values.

The call-by-need calculus we use in this paper was introduced in [2]. It is more concise than previous specifications of call-by-need [5, 4, 32, 13], but it is operationally equivalent to them, so that the results of this paper could also be presented by using other alternative specifications.

Given a countable infinite set of symbols $x, y, z, \ldots$ we define different syntactic categories for terms, values, list contexts, answers and need contexts:

Terms	$t, u ::= x \mid tu \mid \lambda x.t \mid t[x/u]$
Values	$v ::= \lambda x.t$
List contexts	$\mathtt{L} ::= \Box \mid \mathtt{L}[x/t]$
Answers	$\mathtt{A} ::= \mathtt{L}[\lambda y.t]$
Need contexts	$\mathtt{M}, \mathtt{N} ::= \Box \mid \mathtt{N}t \mid \mathtt{N}[x/t] \mid \mathtt{N}[\![x]\!][x/\mathtt{M}]$

Terms of the form $t[x/u]$ are **closures**, and $[x/u]$ is said to be an **explicit substitution** (ES). The set of all the terms is denoted by $\mathcal{T}_{\mathtt{e}}$; remark that $\mathcal{T}_{\mathtt{a}} \subset \mathcal{T}_{\mathtt{e}}$. The notions of **free** and **bound** variables are defined as expected, in particular, $\mathtt{fv}(t[x/u]) := \mathtt{fv}(t)\backslash\{x\} \cup \mathtt{fv}(u)$, $\mathtt{fv}(\lambda x.t) := \mathtt{fv}(t)\backslash\{x\}$, $\mathtt{bv}(t[x/u]) := \mathtt{bv}(t) \cup \{x\} \cup \mathtt{bv}(u)$ and $\mathtt{bv}(\lambda x.t) := \mathtt{bv}(t) \cup \{x\}$.

We use $\mathtt{N}[t]$ (resp. $\mathtt{L}[t]$) for the term obtained by replacing the hole of $\mathtt{N}$ (resp. $\mathtt{L}$) by the term t. We use the special notation $\mathtt{N}[\![u]\!]$ or $\mathtt{L}[\![u]\!]$ when the free variables of u are not captured by the context, *i.e.* there are no abstractions or explicit substitutions in the context that binds the free variables of u. Thus for example, given $\mathtt{N} = (\Box x)[x/z]$, we have $(yx)[x/z] = \mathtt{N}[y] = \mathtt{N}[\![y]\!]$, but $(xx)[x/z] = \mathtt{N}[x]$ cannot be written as $\mathtt{N}[\![x]\!]$. Notice the use of this special notation in the last case of needed contexts, an example of such case being $(xy)[y/t][x/\Box]$.

The **call-by-need calculus**, introduced in [2], is given by the set of terms $\mathcal{T}_{\mathtt{e}}$ and the **reduction relation** $\to_{\mathtt{need}}$, the **union** of $\to_{\mathtt{dB}}$ and $\to_{\mathtt{lsv}}$, which are, respectively, the closure by *need* contexts $\mathtt{N}$ of the following rewriting rules:

$$\begin{array}{ll} \mathtt{L}[\lambda x.t]u & \mapsto_{\mathtt{dB}} \mathtt{L}[t[x/u]] \\ \mathtt{N}[\![x]\!][x/\mathtt{L}[v]] & \mapsto_{\mathtt{lsv}} \mathtt{L}[\mathtt{N}[\![v]\!][x/v]] \end{array}$$

An example of `need`-reduction sequence is

$$\begin{array}{llll} (\lambda x_1.\mathtt{I}(x_1\mathtt{I}))(\lambda y.\mathtt{I}y) & \to_{\mathtt{dB}} & (\mathtt{I}(x_1\mathtt{I}))[x_1/\lambda y.\mathtt{I}y] & \to_{\mathtt{dB}} \\ x_2[x_2/x_1\mathtt{I}][x_1/\lambda y.\mathtt{I}y] & \to_{\mathtt{lsv}} & x_2[x_2/(\lambda x_3.\mathtt{I}x_3)\mathtt{I}][x_1/\lambda y.\mathtt{I}y] & \to_{\mathtt{dB}} \\ x_2[x_2/(\mathtt{I}x_3)[x_3/\mathtt{I}]][x_1/\lambda y.\mathtt{I}y] & \to_{\mathtt{dB}} & x_2[x_2/x_4[x_4/x_3][x_3/\mathtt{I}]][x_1/\lambda y.\mathtt{I}y] & \to_{\mathtt{lsv}} \\ x_2[x_2/x_4[x_4/\mathtt{I}][x_3/\mathtt{I}]][x_1/\lambda y.\mathtt{I}y] & \to_{\mathtt{lsv}} & x_2[x_2/\mathtt{I}[x_4/\mathtt{I}][x_3/\mathtt{I}]][x_1/\lambda y.\mathtt{I}y] & \to_{\mathtt{lsv}} \\ \mathtt{I}[x_2/\mathtt{I}][x_4/\mathtt{I}][x_3/\mathtt{I}][x_1/\lambda y.\mathtt{I}y] & & & \end{array}$$

As for call-by-name, reduction preserves free variables, *i.e.* $t \to_{\mathtt{need}} t'$ implies $\mathtt{fv}(t) \supseteq \mathtt{fv}(t')$. Notice that call-by-need reduction is also weak, so that answers are not `need`-reducible. Among the free variables of terms we distinguish the set of **needed free variables**, defined as follows:

$$\begin{array}{lll} \mathtt{nv}(x) := \{x\} & \mathtt{nv}(tu) := \mathtt{nv}(t) & \\ \mathtt{nv}(\lambda x.t) := \emptyset & \mathtt{nv}(t[y/u]) := \begin{cases} (\mathtt{nv}(t) \setminus \{y\}) \cup \mathtt{nv}(u) & \text{if } y \in \mathtt{nv}(t) \\ \mathtt{nv}(t) \setminus \{y\} & \text{if } y \notin \mathtt{nv}(t) \end{cases} & \end{array}$$

A useful property about needed free variables can be stated as follows:

Lemma 1. *Let $t \in \mathcal{T}_{\mathtt{e}}$. Then $x \in \mathtt{nv}(t)$ if and only if there exists $\mathtt{N}$ s.t. $t = \mathtt{N}[\![x]\!]$.*

Thus for example, for $t = x_1[x_1/x_2y][x_2/x_3y']$, we have $x_3 \in \mathtt{nv}(t)$ and $t = \mathtt{N}[\![x_3]\!]$, for $\mathtt{N} = x_1[x_1/x_2y][x_2/\Box y']$.

We now need some general notions applicable to any reduction system $\mathcal{R}$. We denote by $\twoheadrightarrow_{\mathcal{R}}$ (resp. $\to^+_{\mathcal{R}}$) the **reflexive-transitive** (resp. **transitive**) closure of any given reduction relation $\to_{\mathcal{R}}$. A term t is **in $\mathcal{R}$-normal form**, written $t \in \mathcal{R}$-nf, if there is no t' s.t. $t \to_{\mathcal{R}} t'$; and t **has an $\mathcal{R}$-normal form** iff there is $t' \in \mathcal{R}$-nf such that $t \twoheadrightarrow_{\mathcal{R}} t'$. A term t is said to be **weakly $\mathcal{R}$-normalizing**,

or $\mathcal{R}$**-normalizing**, written $t \in \mathcal{WN}(\mathcal{R})$, iff t has an $\mathcal{R}$-nf.

It is well-known that the set of $\mathcal{T}_{\mathtt{a}}$-terms that are in `name`-nf can be characterized by the following grammar:

$$\mathtt{Na} ::= \lambda x.t \mid \mathtt{Nva} \qquad\qquad \mathtt{Nva} ::= x \mid \mathtt{Nva}\ t$$

Similarly, $\mathcal{T}_{\mathtt{e}}$-terms in `need`-nf can be specified by the following grammar:

$$\begin{array}{l}\mathtt{Ne} \ ::= \mathtt{L}[\lambda x.t] \mid \mathtt{Nae} \\ \mathtt{Nae} ::= x \mid \mathtt{Nae}\ t \mid \mathtt{Nae}[x/u]\ x \notin \mathtt{nv}(\mathtt{Nae}) \mid \mathtt{Nae}[x/\mathtt{Nae}]\ x \in \mathtt{nv}(\mathtt{Nae})\end{array}$$

Lemma 2. *Let $t \in \mathcal{T}_{\mathtt{e}}$. Then $t \in \mathtt{Ne}$ iff t is in* `need`*-nf.*

Proof. The proof proceeds by first showing that $t \in \mathtt{Nae}$ iff t is in `need`-nf and t is not an answer. Then the statement of the lemma follows. ◇

In order to relate `need`-nfs with `name`-nfs we use the following function $\_^{\downarrow}$ defined on $\mathcal{T}_{\mathtt{e}}$-terms.

$$\begin{array}{ll} x^{\downarrow} := x & (\lambda x.t)^{\downarrow} := \lambda x.t^{\downarrow} \\ (tu)^{\downarrow} := t^{\downarrow}u^{\downarrow} & t[x/u]^{\downarrow} := t^{\downarrow}\{x/u^{\downarrow}\}\end{array}$$

Notice that $x \in \mathtt{nv}(t)$ implies $x \in \mathtt{nv}(t^{\downarrow})$.

Lemma 3. *Let $t \in \mathcal{T}_{\mathtt{e}}$. Then t in* `need`*-nf implies $t^{\downarrow}$ in* `name`*-nf.*

Proof. We first remark that $t\{x/u\} \in \mathtt{Nva}$ iff $t \in \mathtt{Nva}$ and $x \notin \mathtt{nv}(t)$, or t and u are in $\mathtt{Nva}$ and $x \in \mathtt{nv}(t)$. The proof then proceeds by showing the following two statements by induction on the grammars.

$$1)\ t \in \mathtt{Nae} \text{ implies } t^{\downarrow} \in \mathtt{Nva} \qquad\qquad 2)\ t \in \mathtt{Ne} \text{ implies } t^{\downarrow} \in \mathtt{Na}$$

1) a). If $t = x \in \mathtt{Nae}$, then the statement is straightforward. b). If $t = t_1 t_2 \in \mathtt{Nae}$, where $t_1 \in \mathtt{Nae}$, then the *i.h.* gives ${t_1}^{\downarrow} \in \mathtt{Nva}$ so that $t^{\downarrow} = {t_1}^{\downarrow}{t_2}^{\downarrow} \in \mathtt{Nva}$. c). If $t = t_1[x/t_2] \in \mathtt{Nae}$, where $t_1 \in \mathtt{Nae}$, then the *i.h.* gives ${t_1}^{\downarrow} \in \mathtt{Nva}$. We distinguish two cases. If $x \notin \mathtt{nv}(t_1)$, then $t^{\downarrow} = {t_1}^{\downarrow}\{x/{t_2}^{\downarrow}\} \in \mathtt{Nva}$. If $x \in \mathtt{nv}(t_1)$, then necessarily $t_2 \in \mathtt{Nva}$, so that the *i.h.* gives ${t_2}^{\downarrow} \in \mathtt{Nva}$. Then, $t^{\downarrow} = {t_1}^{\downarrow}\{x/{t_2}^{\downarrow}\} \in \mathtt{Nva}$.
2) a). If $t = \mathtt{L}[\lambda x.u] \in \mathtt{Ne}$, then $t^{\downarrow}$ is necessarily an abstraction so that $t^{\downarrow} \in \mathtt{Na}$. b). If $t \in \mathtt{Ne}$ comes from $t \in \mathtt{Nae}$, then we apply the previous point. ◇

3 Non-idempotent Intersection Types

Our results rely on typability of terms in suitable systems with non-idempotent intersection types. This section introduces such type systems. The first one, called $\mathcal{V}$ (for values), characterizes the set of `name`-normalizing terms; it extends the one in [19] which characterizes the set of *head* normalizing terms in λ-calculus.

The second one, called $\mathcal{A}$ (for answers), characterizes the set of `need`-normalizing terms; it extends the one in [24] which characterizes the set of *head-linear* normalizing terms in λ-calculus with explicit substitutions.

In order to introduce the type systems, let us first introduce the notion of type. We denote finite multisets by double curly brackets, so that $\{\!\{\,\}\!\}$ denotes the empty multiset; $\{\!\{a, a, b\}\!\}$ denotes a multiset having two occurrences of the element a and one occurrence of b. We use $\sqcup$ for multiset union. Given a constant type $\mathtt{a}$ which denotes *answers*, and a countable infinite set of *base types* $\alpha, \beta, \gamma, \ldots$, we consider **types** and **multiset types** defined by the following grammars:

Types	τ, σ, ρ	$::= \mathtt{a} \mid \alpha \mid \mathcal{M} \to \tau$	
Multiset types	$\mathcal{M}$	$::= \{\!\{\tau_i\}\!\}_{i \in I}$	where I is a finite set

The type $\{\!\{\,\}\!\}$ plays the rôle of the universal ω type in [14, 12]. Remark that types are *strict* [16, 37], *i.e.* the right-hand sides of functional types are never multisets. Moreover, they make use of standard notations for multisets, as in [19], so that $\{\!\{\sigma, \sigma, \tau\}\!\}$ needs to be understood as the intersection type $\sigma \wedge \sigma \wedge \tau$, where the symbol $\wedge$ is commutative and associative but **non-idempotent**, *i.e.* $\sigma \wedge \sigma$ and σ are not equivalent.

Type assignments, written Γ, Δ, are functions from variables to multiset types, assigning the empty multiset to all but a finite set of the variables. The domain of Γ is $\mathtt{dom}(\Gamma) := \{x \mid \Gamma(x) \neq \{\!\{\,\}\!\}\}$. The **union of type assignments**, written $\Gamma + \Delta$, is a type assignment defined by $(\Gamma + \Delta)(x) := \Gamma(x) \sqcup \Delta(x)$, where $\sqcup$ denotes multiset union; thus, $\mathtt{dom}(\Gamma + \Delta) = \mathtt{dom}(\Gamma) \cup \mathtt{dom}(\Delta)$. An example is $(x{:}\{\!\{\sigma\}\!\}, y{:}\{\!\{\tau\}\!\}) + (x{:}\{\!\{\sigma'\}\!\}, z{:}\{\!\{\tau'\}\!\}) = (x{:}\{\!\{\sigma, \sigma'\}\!\}, y{:}\{\!\{\tau\}\!\}, z{:}\{\!\{\tau'\}\!\})$. For convenience, we write $+_{i \in I} \Gamma_i$ for a finite union of type assignments (where $I = \emptyset$ gives an empty function), instead of the more traditional notation $\Sigma_{i \in I} \Gamma_i$. When $\mathtt{dom}(\Gamma)$ and $\mathtt{dom}(\Delta)$ are disjoint we use $\Gamma; \Delta$ instead of $\Gamma + \Delta$, and we write $x{:}\{\!\{\sigma_i\}\!\}_{i \in I}; \Gamma$, even when $I = \emptyset$, for the assignment $(x{:}\{\!\{\sigma_i\}\!\}_{i \in I}; \Gamma)(x) = \{\!\{\sigma_i\}\!\}_{i \in I}$ and $(x{:}\{\!\{\sigma_i\}\!\}_{i \in I}; \Gamma)(y) = \Gamma(y)$ if $y \neq x$. We denote by $\Gamma \setminus\!\setminus x$ the assignment $(\Gamma \setminus\!\setminus x)(x) = \{\!\{\,\}\!\}$ and $(\Gamma \setminus\!\setminus x)(y) = \Gamma(y)$ if $y \neq x$. We write $x \# \Gamma$ iff $x \notin \mathtt{dom}(\Gamma)$.

Type judgments have the form $\Gamma \vdash t{:}\tau$, where Γ is a type assignment, t is a term and τ is a type. The **$\mathcal{V}$-type system** ($\mathcal{V}$ for value) for $\mathcal{T}_{\mathtt{a}}$-terms is given by the typing rules in Fig. 1 and the **$\mathcal{A}$-type system** ($\mathcal{A}$ for answer) for $\mathcal{T}_{\mathtt{e}}$-terms is given by the typing rules in Fig. 2. System $\mathcal{V}$ can be seen as the restriction of system $\mathcal{A}$ to $\mathcal{T}_{\mathtt{a}}$-terms. Given $\mathcal{S} \in \{\mathcal{V}, \mathcal{A}\}$, a **(type) derivation** in $\mathcal{S}$ is a tree obtained by applying the (inductive) rules of the system $\mathcal{S}$. We write $\Phi \rhd_{\mathcal{S}} \Gamma \vdash t{:}\tau$ if there is a derivation Φ in system $\mathcal{S}$ typing t, *i.e.* ending in the type judgment $\Gamma \vdash t{:}\tau$. A term t is **typable** in $\mathcal{S}$ iff there is a derivation in $\mathcal{S}$ typing t. The **size** of a type derivation Φ is a natural number $\mathtt{sz}(\Phi)$ denoting the number of nodes of the tree Φ.

The constant type $\mathtt{a}$ in rules (`val`) and (`ans`) is used to type values and answers respectively, and rule (`cut`) of system $\mathcal{A}$ may also type answers (when $\tau = \mathtt{a}$ and $t = \mathtt{L}[\lambda x.t']$). The axiom (`ax`) is *relevant* (there is no weakening) and the rules ($\to$ `e`) and (`cut`) are *multiplicative*; both characteristics being related to the *resource aware* semantics of the underlying calculus. A particular case of

$$\frac{}{x{:}\{\!\{\tau\}\!\} \vdash x{:}\tau}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash t{:}\{\!\{\sigma_i\}\!\}_{i\in I} \to \tau \qquad (\Delta_i \vdash u{:}\sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash tu{:}\tau}\ (\to \mathtt{e})$$

$$\frac{\Gamma \vdash t{:}\tau}{\Gamma \setminus\!\setminus x \vdash \lambda x.t{:}\Gamma(x) \to \tau}\ (\to \mathtt{i}) \qquad \frac{}{\vdash \lambda x.t{:}\mathtt{a}}\ (\mathtt{val})$$

Fig. 1. The non-idempotent intersection type system $\mathcal{V}$ for terms in $\mathcal{T}_{\mathtt{a}}$

$$\frac{}{x{:}\{\!\{\tau\}\!\} \vdash x{:}\tau}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash t{:}\{\!\{\sigma_i\}\!\}_{i\in I} \to \tau \qquad (\Delta_i \vdash u{:}\sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash tu{:}\tau}\ (\to \mathtt{e})$$

$$\frac{\Gamma \vdash t{:}\tau}{\Gamma \setminus\!\setminus x \vdash \lambda x.t{:}\Gamma(x) \to \tau}\ (\to \mathtt{i}) \qquad \frac{}{\vdash \mathtt{L}[\lambda x.t]{:}\mathtt{a}}\ (\mathtt{ans})$$

$$\frac{x{:}\{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma \vdash t{:}\tau \qquad (\Delta_i \vdash u{:}\sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash t[x/u]{:}\tau}\ (\mathtt{cut})$$

Fig. 2. The non-idempotent intersection type system $\mathcal{A}$ for terms in $\mathcal{T}_{\mathtt{e}}$

rule $(\to \mathtt{e})$ (resp. $(\mathtt{cut})$) is when $I = \emptyset$: the subterm u occurring in the *typed* term tu (resp $t[x/u]$) turns out to be *untyped*. Thus for example, if Ω is the (untypable) non-terminating term $(\lambda z.zz)(\lambda z.zz)$, then from the type derivation of $x{:}\{\!\{\sigma\}\!\} \vdash \lambda y.x{:}\{\!\{\ \}\!\} \to \sigma$ we can construct one for $x{:}\{\!\{\sigma\}\!\} \vdash (\lambda y.x)\Omega{:}\sigma$, and from the type derivation of $x{:}\{\!\{\sigma\}\!\} \vdash x{:}\sigma$ we can construct one for $x{:}\{\!\{\sigma\}\!\} \vdash x[y/\Omega]{:}\sigma$. A major difference between system $\mathcal{V}$ and the one in [19] is the rule $(\mathtt{val})$ which types *any* kind of abstraction. Indeed, given $\Delta = \lambda x.xx$, the abstraction $\lambda x.\Delta\Delta$ –not being head-normalizing in λ-calculus– is typable in our type system (which characterizes `name`-normalizing terms) but is not typable in [19] (which characterizes head-normalizing terms). The same remark applies to the rule $(\mathtt{ans})$ in system $\mathcal{A}$ with respect to the one in [24]. Here is an example of typing derivation in system $\mathcal{A}$.

$$\dfrac{\dfrac{\dfrac{z : \{\!\{\{\!\{\sigma\}\!\} \to \sigma\}\!\} \vdash z{:}\{\!\{\sigma\}\!\} \to \sigma \qquad x : \{\!\{\{\!\{\sigma\}\!\} \to \sigma\}\!\} \vdash x{:}\{\!\{\sigma\}\!\} \to \sigma}{x : \{\!\{\{\!\{\sigma\}\!\} \to \sigma\}\!\} \vdash z[z/x]{:}\{\!\{\sigma\}\!\} \to \sigma} \qquad x : \{\!\{\sigma\}\!\} \vdash x{:}\sigma}{x : \{\!\{\sigma, \{\!\{\sigma\}\!\} \to \sigma\}\!\} \vdash z[z/x]x{:}\sigma}}{\vdash \lambda x.z[z/x]x{:}\{\!\{\sigma, \{\!\{\sigma\}\!\} \to \sigma\}\!\} \to \sigma}$$

It is worth noticing that system $\mathcal{A}$ is a conservative extension of system $\mathcal{V}$:

Lemma 4. *Let $t \in \mathcal{T}_{\mathtt{a}}$. Then $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t{:}\sigma$ iff $\Phi \rhd_{\mathcal{A}} \Gamma \vdash t{:}\sigma$.*

A last remark of this section concerns relevance. Indeed, given $\Phi \rhd_{\mathcal{S}} \Gamma \vdash t{:}\sigma$, not every free variable in t necessarily appears in the domain of Γ. More precisely, both systems enjoy the following property, that can be easily shown by induction on derivations.

Lemma 5. *Let $\mathcal{S} \in \{\mathcal{V}, \mathcal{A}\}$. If $\Phi \rhd_{\mathcal{S}} \Gamma \vdash t{:}\sigma$ then $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

4 Characterization of name-normalizing terms

In this section we adapt the (standard) characterization of head-normalizing terms of the λ-calculus to our call-by-name calculus, *i.e.* we show that a term t is name-normalizing iff t is typable in system $\mathcal{V}$.

The characterization is split in two parts. The first one shows that typable terms in system $\mathcal{V}$ are name-normalizing, a result which is based on a *weighted subject reduction* property. In contrast to similar results for *idempotent* intersection type systems, which are typically based on *reducibility* arguments, quantitative information of type derivations, obtained by means of *non-idempotent* types, provides a completely *combinatorial* proof. This is exactly the place where the quantitative approach by means of non-idempotent types makes the difference. The second part of the characterization shows that name-normalizing terms are typable in $\mathcal{V}$, and is based on a *subject expansion* property.

We start with the subject reduction property which uses the following lemma.

Lemma 6 (Substitution). *If $\Phi_t \rhd_{\mathcal{V}} x{:}\{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma \vdash t{:}\tau$ and $(\Phi_u^i \rhd_{\mathcal{V}} \Delta_i \vdash u{:}\sigma_i)_{i\in I}$ then there exists a derivation $\Phi_{t\{x/u\}}$ s.t. $\Phi_{t\{x/u\}} \rhd_{\mathcal{V}} \Gamma +_{i\in I} \Delta_i \vdash t\{x/u\}{:}\tau$. Moreover, $\mathtt{sz}(\Phi_{t\{x/u\}}) = \mathtt{sz}(\Phi_t) +_{i\in I} \mathtt{sz}(\Phi_u^i) - |I|$.*

Proof. By induction on Φ_t. Let $\Phi_t \rhd_{\mathcal{V}} x{:}\{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma \vdash \lambda y.u{:}\mathtt{a}$ so that $\Gamma = \emptyset$ and $I = \emptyset$. Since $\Phi_{t\{x/u\}} \rhd_{\mathcal{V}} \emptyset \vdash \lambda y.u\{x/u\}{:}\mathtt{a}$, we conclude because $\emptyset = \Gamma +_{i\in\emptyset} \Delta_i$. The other cases are similar to those of the λ-calculus (see for example [10]). $\diamond$

Subject reduction can now be stated in the following form.

Theorem 1 (Weighted Subject Reduction for name). *Let $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t{:}\tau$. If $t \to_{\mathtt{name}} t'$, then there exists Φ' s.t. $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash t'{:}\tau$. Moreover, $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.*

Proof. By induction on Φ. The proof proceeds as that of the λ-calculus [10]. It is worth noticing that when $t = (\lambda x.u)u'$, the subterm $\lambda x.u$ is necessarily typed in Φ using rule $(\to \mathtt{i})$, so that Lem. 6 can be applied to conclude. $\diamond$

In order to show that name-normalizing terms are $\mathcal{V}$-typable we use a subject expansion property, which needs the following lemma.

Lemma 7 (Reverse Substitution). *If $\Phi_{t\{x/u\}} \rhd_{\mathcal{V}} \Gamma \vdash t\{x/u\}{:}\tau$, then $\exists \Gamma_0$, $\exists \Phi_t$, $\exists I$, $\exists (\Delta_i)_{i\in I}$, $\exists (\sigma_i)_{i\in I}$, $\exists (\Phi_u^i)_{i\in I}$ such that $\Gamma = \Gamma_0 +_{i\in I} \Delta_i$, and $\Phi_t \rhd_{\mathcal{V}} x : \{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma_0 \vdash t{:}\tau$ and $(\Phi_u^i \rhd_{\mathcal{V}} \Delta_i \vdash u{:}\sigma_i)_{i\in I}$.*

Proof. By induction on $\Phi_{t\{x/u\}}$. Suppose $\Phi_{t\{x/u\}} \rhd_{\mathcal{V}} \emptyset \vdash \lambda y.u{:}\mathtt{a}$ so that $\Gamma = \emptyset$. There are two cases to consider: (1) $t = \lambda y.v$ and $u = v\{x/u\}$. Then $\Phi_t \rhd_{\mathcal{V}} \emptyset \vdash \lambda y.v{:}\mathtt{a}$, where $\Gamma_0 = \emptyset$ and $I = \emptyset$ so that the property trivially holds. (2) $t = x$ and $u = \lambda y.v$. Then $\Phi_t \rhd_{\mathcal{V}} x : \{\!\{\mathtt{a}\}\!\} \vdash x{:}\mathtt{a}$, $\Phi_u \rhd_{\mathcal{V}} \vdash \lambda y.v{:}\mathtt{a}$, so that we let $\Gamma_0 = \emptyset$, $I = \{1\}$, $\sigma_1 = \mathtt{a}$ and $\Delta_1 = \emptyset$ and $\Gamma = \emptyset = \Gamma_0 + \Delta_1$.

All the other cases proceed as the one of the λ-calculus [10]. $\diamond$

Theorem 2 (Subject Expansion for name). *Let $\Phi' \rhd_{\mathcal{V}} \Gamma \vdash t':\tau$. If $t \to_{\mathtt{name}} t'$, then there exists Φ s.t. $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t:\tau$.*

Proof. By induction on Φ' using Lem. 7. ◇

We can now state the following full characterization result of this section:

Theorem 3. *Let $t \in \mathcal{T}_{\mathtt{a}}$. Then, $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t:\tau$ iff $t \in \mathcal{WN}(\mathtt{name})$.*

Proof. Let $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t:\tau$. Then name-reduction necessarily terminates by Thm. 1 and thus $t \in \mathcal{WN}(\mathtt{name})$.

Let $t \in \mathcal{WN}(\mathtt{name})$ so that $t \twoheadrightarrow^{k}_{\mathtt{name}} t'$, where $t' \in$ name-nf. We proceed by induction on k. If $k = 0$ (*i.e.* $t = t'$), then t is a name-nf. We have two cases.

- If $t = \lambda y.v$, then the property trivially holds with $\Gamma = \emptyset$ and $\tau = \mathtt{a}$.
- If $t = xt_1 \dots t_n$ $(n \geq 0)$, let $\Gamma = \{x:\{\!\{\tau\}\!\}\}$, where $\tau = \{\!\{\ \}\!\} \to \cdots \to \{\!\{\ \}\!\} \to \alpha$ (α is any base type and τ contains n occurrences of $\{\!\{\ \}\!\}$). By the rule (ax) and n applications of $(\to \mathtt{e})$ we obtain a derivation in $\mathcal{V}$ ending in $\Gamma \vdash xt_1 \dots t_n:\alpha$.

Otherwise, let $t \to_{\mathtt{name}} u \twoheadrightarrow^{k}_{\mathtt{name}} t'$. By the *i.h.* we have a derivation in $\mathcal{V}$ ending in $\Gamma \vdash u:\tau$. Thus by Thm. 2 the same holds form t. ◇

5 Characterization of need-normalizing terms

This section gives a characterization of need-normalizing terms by means of $\mathcal{A}$-typability, *i.e.* we show that a term t is need-normalizing iff t is typable in system $\mathcal{A}$. At first sight, this result follows the same lines of Sec. 4, however, it is much trickier and requires the development of special tools dealing with need contexts. To the best of our knowledge, this is the first time that call-by-need normalization is characterized by means of typing technology.

Similarly to the call-by-name case, we split the characterization proof in two parts and we use subject reduction and expansion properties to show them. More precisely, the first part of the characterization states that typable terms in system $\mathcal{A}$ are normalizing in call-by-need, a result based on a *weighted subject reduction* property, obtained by using quantitative information of the (non-idempotent) type derivations of system $\mathcal{A}$. The second part of the characterization states that need-normalizing terms are typable in system $\mathcal{A}$, a property based on the *subject expansion* property.

In contrast to the call-by-name case, handling need-reduction is quite involved, particularly due to the use of contexts in the rewriting rules defining the relation. To deal with this difficulty, we introduce a technical tool which consists in extending system $\mathcal{A}$ (*cf.* Sec. 3) with typing rules for *list contexts* (Fig. 3). The type judgments have the form $\Gamma \Vdash \mathtt{L} \rhd \Delta$, where Γ, Δ are type assignments and $\mathtt{L}$ is a list context. The left-hand side Γ of a judgment $\Gamma \Vdash \mathtt{L} \rhd \Delta$ is a type assignment for the (typed) free variables of $\mathtt{L}$, while the right-hand side Δ is a type assignment for the term which is supposed to fill in the hole of $\mathtt{L}$. These

$$\frac{}{\emptyset \Vdash \Box \triangleright \emptyset} \qquad \frac{x : \{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma \Vdash \mathtt{L} \triangleright \Delta \quad x\#\Delta \quad (\Delta_k \vdash u{:}\sigma_k)_{k\in I\uplus J}}{\Gamma +_{k\in I\uplus J} \Delta_k \Vdash \mathtt{L}[x/u] \triangleright \Delta; x : \{\!\{\sigma_j\}\!\}_{j\in J}}$$

Fig. 3. Extending the intersection type system $\mathcal{A}$ to list contexts

rules should then be considered modulo α-conversion. Notice the use of two different kinds of assignments in the second type rule : the assignments $(\Delta_i)_{i\in I}$ are used to type the copies of u affecting the free occurrences of x in the list $\mathtt{L}$, while $(\Delta_j)_{j\in J}$ are used to type the copies of u affecting the free occurrences of x in the term which will fill the hole of $\mathtt{L}$.

The following lemma decomposes a type derivation of a term $\mathtt{L}[t]$ into one derivation for the context $\mathtt{L}$ and another one for the term t. Reciprocally, context and term derivations can be combined if their types coincide. The proof can be done by induction on $\mathtt{L}$.

Lemma 8. *$\Phi_{\mathtt{L}[t]} \rhd_{\mathcal{A}} \Lambda \vdash \mathtt{L}[t] : \sigma$ iff $\exists\Gamma$, $\exists\Pi$, $\exists\Delta$, $\exists\Phi_{\mathtt{L}}$, $\exists\Phi_t$, such that $\Lambda = \Gamma + \Pi$ and $\Phi_{\mathtt{L}} \rhd_{\mathcal{A}} \Gamma \Vdash \mathtt{L} \triangleright \Delta$ and $\Phi_t \rhd_{\mathcal{A}} \Delta; \Pi \vdash t : \sigma$. Moreover, $\mathtt{sz}(\Phi_{\mathtt{L}[t]}) = \mathtt{sz}(\Phi_{\mathtt{L}}) + \mathtt{sz}(\Phi_t) - 1$.*

Combining different derivable typing judgments of the same list context by means of multiset union yields a derivable typing judgment. Moreover, their sizes can be related using the notion of **height**, defined on list contexts as follows: $\mathtt{height}(\Box) := 1$ and $\mathtt{height}(\mathtt{L}'[y/v]) := \mathtt{height}(\mathtt{L}') + 1$.

Lemma 9. *If $(\Phi^j_{\mathtt{L}} \rhd_{\mathcal{A}} \Gamma_j \Vdash \mathtt{L} \triangleright \Delta_j)_{j\in J}$, then there exists a derivation $\Phi_{\mathtt{L}}$ s.t. $\Phi_{\mathtt{L}} \rhd_{\mathcal{A}} +_{j\in J}\Gamma_j \Vdash \mathtt{L} \triangleright +_{j\in J}\Delta_j$. Moreover, $\mathtt{sz}(\Phi_{\mathtt{L}}) = +_{j\in J}\mathtt{sz}(\Phi^j_{\mathtt{L}}) - (\mathtt{height}(\mathtt{L}) \cdot (|J| - 1))$.*

Proof. Notice that the statement also holds in the case $J = \emptyset$. The proof is by induction on $\mathtt{L}$ (see the Appendix for details). ◇

To achieve the proof of the Weighted Subject Reduction property for the `need`-calculus, we first need to guarantee soundness of the (partial) substitution operation used in our framework, *i.e.* if $\mathtt{N}[\![x]\!]$ and u are typable, then $\mathtt{N}[\![u]\!]$ is typable too. Moreover, $|K|$ elements of the multiset type $\{\!\{\sigma_i\}\!\}_{i\in I}$ associated to x in the type derivation of $\mathtt{N}[\![x]\!]$ are consumed, for some $K \subseteq I$.

Lemma 10 (Partial Substitution). *If $\Phi_{\mathtt{N}[\![x]\!]} \rhd_{\mathcal{A}} x{:}\{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma \vdash \mathtt{N}[\![x]\!]{:}\tau$ and $(\Phi^i_u \rhd_{\mathcal{A}} \Delta_i \vdash u{:}\sigma_i)_{i\in I}$ then $\exists\Phi_{\mathtt{N}[\![u]\!]}$ s.t. $\Phi_{\mathtt{N}[\![u]\!]} \rhd_{\mathcal{A}} x{:}\{\!\{\sigma_i\}\!\}_{i\in I\setminus K}; \Gamma +_{k\in K} \Delta_k \vdash \mathtt{N}[\![u]\!]{:}\tau$, for some $K \subseteq I$ where $\mathtt{sz}(\Phi_{\mathtt{N}[\![u]\!]}) = \mathtt{sz}(\Phi_{\mathtt{N}[\![x]\!]}) +_{k\in K} \mathtt{sz}(\Phi^k_u) - |K|$.*

Proof. By induction on typing derivations (a similar proof appears in [23]). ◇

To complete the subject reduction argument we still need to guarantee that every needed variable of a typed term is necessarily typed, a property which is specified by means of the following lemma.

Lemma 11. *If $\Phi \rhd_{\mathcal{A}} \Gamma \vdash \mathtt{N}[\![x]\!]{:}\tau$, then $\exists \Gamma'$, $\exists I \neq \emptyset$, and $\exists (\sigma_i)_{i\in I}$ such that $\Gamma = x : \{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma'$.*

Proof. By induction on $\mathtt{N}$. The base case $\mathtt{N} = \Box$ is straightforward. We only show here the interesting inductive case $\mathtt{N}[\![x]\!] = \mathtt{N}_1[\![y]\!][y/\mathtt{N}_2[\![x]\!]]$, for which Φ has necessarily the following form, where $\Gamma = \Pi +_{j\in J} \Delta_j$.

$$\frac{y : \{\!\{\rho_j\}\!\}_{j\in J}; \Pi \vdash \mathtt{N}_1[\![y]\!]{:}\tau \qquad (\Delta_j \vdash \mathtt{N}_2[\![x]\!]{:}\rho_j)_{j\in J}}{\Pi +_{j\in J} \Delta_j \vdash \mathtt{N}_1[\![y]\!][y/\mathtt{N}_2[\![x]\!]]{:}\tau}$$

By the *i.h.* on the left derivation we have $J \neq \emptyset$. By the *i.h.* on the right derivations $\Delta_j = x : \{\!\{\tau_j^i\}\!\}_{i\in I_j}; \Delta'_j$ and $I_j \neq \emptyset$ for all $j \in J$. Moreover, $\Pi = x : \{\!\{\tau_k\}\!\}_{k\in K}; \Pi'$, where K could be empty. Then

$$\begin{array}{l} \Pi +_{j\in J} \Delta_j = (x : \{\!\{\tau_k\}\!\}_{k\in K}; \Pi') +_{j\in J} (x : \{\!\{\tau_j^i\}\!\}_{i\in I_j}; \Delta'_j) = \\ x : \{\!\{\tau_k\}\!\}_{k\in K} +_{j\in J} x : \{\!\{\tau_j^i\}\!\}_{i\in I_j}; (\Pi' +_{j\in J} \Delta'_j) \end{array}$$

The property then holds for $\{\!\{\sigma_i\}\!\}_{i\in I} = \{\!\{\tau_k\}\!\}_{k\in K} +_{j\in J} \{\!\{\tau_j^i\}\!\}_{i\in I_j}$ and $\Gamma' = \Pi' +_{j\in J} \Delta'_j$. We have $I \neq \emptyset$ since $J \neq \emptyset$ and $I_j \neq \emptyset$ for all $j \in J$. ◇

Subject reduction can now be stated in a combinatorial way, by using the $\mathtt{sz}(\_)$ measure on derivations defined in Sec. 3.

Theorem 4 (Weighted Subject Reduction for need). *Let $\Phi \rhd_{\mathcal{A}} \Gamma \vdash t{:}\tau$. If $t \to_{\mathtt{need}} t'$, then there exists Φ' s.t. $\Phi' \rhd_{\mathcal{A}} \Gamma \vdash t'{:}\tau$. Moreover, $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.*

Proof. By induction on $t \to_{\mathtt{need}} t'$, using Lem. 8, Lem. 9, Lem. 10, and Lem. 11. We refer the interested reader to the Appendix for full details of the proof. ◇

To show that need-normalizing terms are $\mathcal{A}$-typable we use a subject expansion property, which needs the following lemma.

Lemma 12 (Reverse Partial Substitution). *Let $\mathtt{N}[\![x]\!], s$ be terms s.t. $x \notin \mathtt{fv}(s)$ and $\Phi_{\mathtt{N}[\![s]\!]} \rhd_{\mathcal{A}} \Gamma \vdash \mathtt{N}[\![s]\!]{:}\tau$. Then $\exists \Gamma_0$, $\exists \Phi_{\mathtt{N}[\![x]\!]}$, $\exists I$, $\exists (\Delta_i)_{i\in I}$, $\exists (\sigma_i)_{i\in I}$, $\exists (\Phi_s^i)_{i\in I}$ s.t. $\Gamma = \Gamma_0 +_{i\in I} \Delta_i$ and $\Phi_{\mathtt{N}[\![x]\!]} \rhd_{\mathcal{A}} x : \{\!\{\sigma_i\}\!\}_{i\in I} + \Gamma_0 \vdash \mathtt{N}[\![x]\!]{:}\tau$ and $(\Phi_s^i \rhd_{\mathcal{A}} \Delta_i \vdash s{:}\sigma_i)_{i\in I}$.*

Proof. By induction on the structure of $\mathtt{N}[\![s]\!]$.

- If $\mathtt{N} = \Box$, then $\mathtt{N}[\![s]\!] = s$ and the result holds, for $\Gamma_0 = \emptyset$, $|I| = 1$ and $\sigma_1 = \tau$.
- If $\mathtt{N} = \lambda y.\mathtt{M}$ then the property is straightforward by the *i.h.* (since $y \notin \mathtt{fv}(s)$ by α-conversion).
- If $\mathtt{N} = \mathtt{M}r$ then $\mathtt{N}[\![s]\!] = \mathtt{M}[\![s]\!]r$ and by construction $\Gamma = \Pi +_{j\in J} \Gamma_j$ and $\Phi_{\mathtt{M}[\![s]\!]} \rhd \Pi \vdash \mathtt{M}[\![s]\!]{:}\{\!\{\rho_j\}\!\}_{j\in J} \to \tau$ and $(\Phi_r^j \rhd \Gamma_j \vdash r{:}\rho_j)_{j\in J}$. By the *i.h.* $\Pi = \Pi_0 +_{i\in I} \Delta_i$ where $x : \{\!\{\sigma_i\}\!\}_{i\in I} + \Pi_0 \vdash \mathtt{M}[\![x]\!]{:}\{\!\{\rho_j\}\!\}_{j\in J} \to \tau$ and $(\Delta_i \vdash s{:}\sigma_i)_{i\in I}$. Then, by the rule $(\to \mathtt{e})$, $x : \{\!\{\sigma_i\}\!\}_{i\in I} + \Pi_0 +_{j\in J} \Gamma_j \vdash \mathtt{M}[\![x]\!]r{:}\tau$. The result then holds for $\Gamma_0 := \Pi_0 +_{j\in J} \Gamma_j$.

– If $\mathtt{N} = r\mathtt{M}$ then $\mathtt{N}[\![s]\!] = r\mathtt{M}[\![s]\!]$ and by construction $\Gamma = \Pi +_{j\in J} \Gamma_j$ and $\Phi_r \rhd \Pi \vdash r{:}\{\!\{\rho_j\}\!\}_{j\in J} \to \tau$ and $(\Phi^j_{\mathtt{M}[\![s]\!]} \rhd \Gamma_j \vdash \mathtt{M}[\![s]\!]{:}\rho_j)_{j\in J}$. By the *i.h.* for each $j \in J$, $\Gamma_j = \Gamma_0^j +_{i\in I_j} \Gamma_i^j$ where $x : \{\!\{\sigma_i\}\!\}_{i\in I_j} + \Gamma_0^j \vdash \mathtt{M}[\![x]\!]{:}\rho_j$ and $(\Gamma_i^j \vdash s{:}\sigma_i)_{i\in I_j}$. Let $I := \cup_{j\in J} I_j$. Then, by the rule $(\to \mathtt{e})$, $\Pi +_{j\in J} (x : \{\!\{\sigma_i\}\!\}_{i\in I_j} + \Gamma_0^j) \vdash r\mathtt{M}[\![x]\!]{:}\tau$. Note that $\Pi +_{j\in J} (x : \{\!\{\sigma_i\}\!\}_{i\in I_j} + \Gamma_0^j) = x : \{\!\{\sigma_i\}\!\}_{i\in I} + \Pi +_{j\in J} \Gamma_0^j$. The result then holds for $\Gamma_0 := \Pi +_{j\in J} \Gamma_0^j$.
– All the remaining cases are similar to the previous ones. ◇

Theorem 5 (Subject Expansion for need). *Let $\Phi' \rhd_{\mathcal{A}} \Gamma \vdash t'{:}\tau$. If $t \to_{\mathtt{need}} t'$, then there exists Φ s.t. $\Phi \rhd_{\mathcal{A}} \Gamma \vdash t{:}\tau$.*

Proof. By induction on Φ' using Lem. 8, Lem. 9 and Lem. 12. ◇

To state the full characterization result of this section, we still need to relate typing of $\mathcal{T}_{\mathtt{e}}$-terms in system $\mathcal{A}$ with typing of $\mathcal{T}_{\mathtt{a}}$-terms in system $\mathcal{V}$.

Lemma 13. *Let $t \in \mathcal{T}_{\mathtt{e}}$. Then $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t^{\downarrow}{:}\sigma$ implies $\Phi' \rhd_{\mathcal{A}} \Gamma \vdash t{:}\sigma$.*

Proof. By induction on t.

– If $t = x$, then $t^{\downarrow} = x$. The derivation Φ has the form $x : \{\!\{\sigma\}\!\} \vdash x{:}\sigma$ so that Φ' has also the form $x : \{\!\{\sigma\}\!\} \vdash x{:}\sigma$.
– If $t = \lambda x.u$, then $t^{\downarrow} = \lambda x.u^{\downarrow}$. We have $\Gamma = \Delta \setminus\!\setminus x$ and $\sigma = \Delta(x) \to \tau$, where $\Delta \setminus\!\setminus x \vdash \lambda x.u^{\downarrow}{:}\Delta(x) \to \tau$ is necessarily derivable from a derivation $\Phi_u \rhd_{\mathcal{V}} \Delta \vdash u^{\downarrow}{:}\tau$. The *i.h.* gives $\Phi'_u \rhd_{\mathcal{A}} \Delta \vdash u{:}\tau$ so that we construct $\Phi' \rhd_{\mathcal{A}} \Delta \setminus\!\setminus x \vdash \lambda x.u{:}\Delta(x) \to \tau$ which concludes the proof.
– If $t = t_1 t_2$, then the proof proceeds by induction as in the previous case.
– If $t = t_1[x/t_2]$, then $t^{\downarrow} = t_1{}^{\downarrow}\{x/t_2{}^{\downarrow}\}$. By Lem. 7 we know that $\Gamma = \Gamma_0 +_{i\in I} \Delta_i$ and $\Phi_{t_1{}^{\downarrow}} \rhd_{\mathcal{V}} x : \{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma_0 \vdash t_1{}^{\downarrow}{:}\tau$ and $(\Phi^i_{t_2{}^{\downarrow}} \rhd_{\mathcal{V}} \Delta_i \vdash t_2{}^{\downarrow}{:}\sigma_i)_{i\in I}$. The *i.h.* then gives $\Phi'_{t_1} \rhd_{\mathcal{A}} x : \{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma_0 \vdash t_1{:}\tau$ and $(\Phi'^i_{t_2} \rhd_{\mathcal{A}} \Delta_i \vdash t_2{:}\sigma_i)_{i\in I}$. We then conclude $\Phi'_t \rhd_{\mathcal{A}} \Gamma \vdash t_1[x/t_2]{:}\tau$ by using rule $(\mathtt{cut})$. ◇

We can now state the full characterization result, which is one of the main results of this paper. The implication "typable terms in system $\mathcal{A}$ are need-normalizing" is obtained through the *weighted subject reduction*, which provides a completely *combinatorial* argument of this property, as in the call-by-name case. This is exactly the advantage provided by the quantitative approach based on non-idempotent types, which makes the difference with a *qualitative* system based on idempotent intersection types.

Theorem 6. *Let $t \in \mathcal{T}_{\mathtt{e}}$. Then, $\Phi \rhd_{\mathcal{A}} \Gamma \vdash t{:}\tau$ iff $t \in \mathcal{WN}(\mathtt{need})$.*

Proof. Let $\Phi \rhd \Gamma \vdash t{:}\tau$. Then need-reduction necessarily terminates by Thm. 4 so that $t \in \mathcal{WN}(\mathtt{need})$.

Let $t \in \mathcal{WN}(\mathtt{need})$, so that $t \twoheadrightarrow^k_{\mathtt{need}} t'$, where $t' \in$ need-nf. We proceed by induction on k. If $k = 0$ (*i.e.* $t = t'$), then t is in need-nf. By Lem. 3 $t^{\downarrow}$ is in name-nf. Then $\Phi \rhd_{\mathcal{V}} \Gamma \vdash t^{\downarrow}{:}\sigma$ by Thm. 3 and $\Phi \rhd_{\mathcal{A}} \Gamma \vdash t{:}\sigma$ by Lem. 13.

Otherwise, let $t \to_{\mathtt{need}} u \to^k_{\mathtt{need}} t'$. By the *i.h.* we have $\Phi' \rhd_{\mathcal{A}} \Gamma \vdash u{:}\tau$, thus by Thm. 5 we obtain $\Phi \rhd_{\mathcal{A}} \Gamma \vdash t{:}\tau$. ◇

6 Soundness and Completeness

This section uses the two characterization results developed in Sec. 4 and 5 to prove soundness and completeness of call-by-need w.r.t call-by-name. More precisely, a call-by-name interpreter stops in a value if and only if the call-by-need interpreter stops in an answer. This implies that call-by-need and call-by-name are observationally equivalent, so that in particular call-by-need turns out to be a correct implementation of call-by-name.

Lemma 14. *Let $t \in \mathcal{T}_{\mathtt{a}}$. Then $t \in \mathcal{WN}(\mathtt{need}) \Leftrightarrow t \in \mathcal{WN}(\mathtt{name})$.*

Proof. Let $t \in \mathcal{WN}(\mathtt{need})$. Then $\Phi \triangleright_{\mathcal{A}} \Gamma \vdash t{:}\tau$ holds by Thm. 6, and $\Phi \triangleright_{\mathcal{V}} \Gamma \vdash t{:}\tau$ holds by Lem. 4, thus we get $t \in \mathcal{WN}(\mathtt{name})$ by Thm. 3. Conversely, let $t \in \mathcal{WN}(\mathtt{name})$. Then $\Phi \triangleright_{\mathcal{V}} \Gamma \vdash t{:}\tau$ holds by Thm. 3, so that $\Phi \triangleright_{\mathcal{A}} \Gamma \vdash t{:}\tau$ holds by Lem. 4, and thus we get $t \in \mathcal{WN}(\mathtt{need})$ by Thm. 6. ◇

Given a reduction relation $\mathcal{R}$ on a term language $\mathcal{T}$, and an associated notion of context for the term language $\mathcal{T}$, we define t to be **observationally equivalent** to u, written $t \cong_{\mathcal{R}} u$, if and only if $\mathtt{C}[t] \in \mathcal{WN}(\mathcal{R}) \Leftrightarrow \mathtt{C}[s] \in \mathcal{WN}(\mathcal{R})$ for every context $\mathtt{C}$. Since we work with two different term languages $\mathcal{T}_{\mathtt{a}}$ and $\mathcal{T}_{\mathtt{e}}$, we first need to introduce their associated notions of contexts $\mathtt{C}$ and $\mathtt{C}'$, which represent, respectively, contexts without and with explicit substitutions.

$$\begin{array}{l} \mathtt{C} \ ::= \Box \mid \lambda x.\mathtt{C} \mid \mathtt{C}\,t \mid t\,\mathtt{C} \\ \mathtt{C}' ::= \Box \mid \lambda x.\mathtt{C}' \mid \mathtt{C}'\,t \mid t\,\mathtt{C}' \mid \mathtt{C}'[x/t] \mid t[x/\mathtt{C}'] \end{array}$$

We can thus conclude with the last main result of the paper.

Corollary 1. *For all terms t and u in $\mathcal{T}_{\mathtt{a}}$, $t \cong_{\mathtt{name}} u$ iff $t \cong_{\mathtt{need}} u$.*

Proof. First of all let us consider the relation $=_{\mathtt{B}}$, which is the least equivalence relation generated by the axiom $(\lambda x.t_1)t_2 = t_1[x/t_2]$. We remark that when t is typable in $\mathcal{A}$, and $t =_{\mathtt{B}} t'$, then t' is typable in $\mathcal{A}$. This means that we can indistinctly quantify over the two sort of contexts $\mathtt{C}$ and $\mathtt{C}'$ previously defined: indeed, from the set of all the contexts $\mathtt{C}$ we can construct the set of all the contexts $\mathtt{C}'$, and vice-versa. The proof of the corollary then proceeds as follows.
Take $t, u \in \mathcal{T}_{\mathtt{a}}$. Then $t \cong_{\mathtt{name}} u$ iff (definition)
$\mathtt{C}[t] \in \mathcal{WN}(\mathtt{name}) \Leftrightarrow \mathtt{C}[u] \in \mathcal{WN}(\mathtt{name})$ for every context $\mathtt{C}$ iff (Thm. 3)
$\mathtt{C}[t]$ is typable in $\mathcal{V} \Leftrightarrow \mathtt{C}[u]$ is typable in $\mathcal{V}$ for every context $\mathtt{C}$ iff (Lem. 4)
$\mathtt{C}[t]$ is typable in $\mathcal{A} \Leftrightarrow \mathtt{C}[u]$ is typable in $\mathcal{A}$ for every context $\mathtt{C}$ iff (Remark)
$\mathtt{C}'[t]$ is typable in $\mathcal{A} \Leftrightarrow \mathtt{C}'[u]$ is typable in $\mathcal{A}$ for every context $\mathtt{C}'$ iff (Thm. 6)
$\mathtt{C}'[t] \in \mathcal{WN}(\mathtt{need}) \Leftrightarrow \mathtt{C}'[u] \in \mathcal{WN}(\mathtt{need})$ for every context $\mathtt{C}'$ iff (definition)
$t \cong_{\mathtt{need}} u$. ◇

7 Conclusion

This paper gives the first full characterization of call-by-need normalization by means of intersection types. This result, together with the full characterization

of call-by-name normalization, provides a new completeness proof for call-by-need, which is only based on logical arguments and does not make use of any complicated notion of rewriting. The use of non-idempotent types allows us to obtain the result in a combinatorial way, by providing quantitative information about reduction sequences, and without resorting to any reducibility argument.

The paper only considers a core language for call-by-need, but it would be interesting to consider other constructors of lazy languages, *e.g.* data constructors, case-expressions and Haskell's seq operators. Moreover, the results in the paper could be extended to the cyclic call-by-need letrec calculus as well as to full (strong) need normal forms (ongoing work).

Last but not least, we would also like to use the ideas in [20] in order to relate call-by-need and needed reduction by means of intersection types.

References

1. B. Accattoli, E. Bonelli, D. Kesner, and C. Lombardi. A nonstandard standardization theorem. *POPL*, ACM 2014.
2. B. Accattoli, P. Barenbaum, and D. Mazza. Distilling abstract machines. *ICFP*, ACM Press, 2014.
3. B. Accattoli and D. Kesner. The structural λ-calculus. *CSL*, pages 381–395, 2010.
4. Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.
5. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. *POPL*, pages 233–246, ACM Press, 1995.
6. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics (revised edition)*, vol. 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, Amsterdan, The Netherlands, 1984.
7. H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Bulletin of Symbolic Logic*, 48:931–940, 1983.
8. A. Bernadet and S. Lengrand. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science*, 9(4), 2013.
9. G. Boudol, P.-L. Curien, and C. Lavatelli. A semantics for lambda calculi with resources. *Mathematical Structures in Computer Science*, 9(4):437–482, 1999.
10. A. Bucciarelli, D. Kesner, and S. Ronchi Della Rocca. The Inhabitation Problem for Non-Idempotent Intersection Types. TCS, LNCS 8705, pages 341-354, 2014.
11. A. Bucciarelli, D. Kesner, and S. Ronchi Della Rocca. Observability for pair pattern calculi. TLCA, LIPIcs, to appear, 2015.
12. F. Cardone and M. Coppo. Two extension of Curry's type inference system. *Logic and Computer Science*, vol. 31 of *APIC Series*, pages 19–75. Academic Press, 1990.
13. S. Chang and M. Felleisen. The call-by-need lambda calculus, revisited. *ESOP*, LNCS 7211, pages 128–147. Springer, 2012.
14. M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Mathematical Logic Quarterly*, 27(2-6):45–58, 1981.
15. M. Coppo and M. Dezani-Ciancaglini. A new type-assignment for lambda terms. *Archiv für Mathematische Logik und Grundlagenforschung*, 19:139–156, 1978.
16. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame, Journal of Formal Logic*, 21:685–693, 1980.

17. O. Danvy and I. Zerny. A synthetic operational account of call-by-need evaluation. *PPDP*, pages 97–108. ACM Press, 2013.
18. E. De Benedetti and S. Ronchi Della Rocca. Bounding normalization time through intersection types. *ITRS 2012*, EPTCS, pages 48–57, 2013.
19. D. de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. Thèse de doctorat, Université Aix-Marseille II, 2007.
20. P. Gardner. *Discovering Needed Reductions Using Type Theory. TACS*, LNCS 789, pages 555–574, 1994.
21. D. Kesner. A theory of explicit substitutions with safe and full composition. *Logical Methods in Computer Science*, 5(3), 2009.
22. D. Kesner and S. Ó Conchúir. Milner's lambda calculus with partial substitutions. Available at `http://www.pps.univ-paris-diderot.fr/~kesner/papers/shortpartial.pdf`, 2008.
23. D. Kesner and D. Ventura. Quantitative types for intuitionistic calculi, 2014. Technical Report available at `https://hal.archives-ouvertes.fr/hal-00980868`.
24. D. Kesner and D. Ventura. Quantitastive types for the linear substitution calculus. *TCS*, LNCS 8705, pages 296-310, 2014.
25. A. J. Kfoury. A linearization of the lambda-calculus and consequences. Technical report, Boston Universsity, 1996.
26. A. J. Kfoury. A Linearization of the Lambda-Calculus and Consequences. *Journal of Logic and Computation*, 10(3):411–436, 2000.
27. A. J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. *Theoretical Computer Science*, 311(1-3):1–70, 2004.
28. J-L. Krivine. *Lambda-calculus, types and models*. Ellis Horwood, 1993.
29. S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189:17–42, 2004.
30. H. Mairson and P. M. Neergaard. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. *ICFP*, pages 138–149, 2004.
31. J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theoretical Computer Science*, 228(1-2):175–210, 1999.
32. J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. *Journal of Functional Programming*, 8(3):275–317, 1998.
33. R. Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). *ENTCS*, 175(3):65–73, 2007.
34. M. Pagani and S. Ronchi Della Rocca. Linearity, non-determinism and solvability. *Fundamenta Informaticae*, 103:358–373, 2010.
35. M. Pagani and S. Ronchi Della Rocca. Solvability in resource lambda-calculus. *FOSSACS*, LNCS 6014, pages 358–373, 2010.
36. G. D. Plotkin. Call-by-Name, Call-by-Value and the lambda-Calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.
37. S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102(1):135–163, 1992.
38. D. Ventura, A. Bucciarelli, and D. Kesner. A combinatorial argument for termination properties. The Brazilian Logic Conference (EBL), 2013.
39. C. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. Phd thesis, Oxford University, 1971.

8 Appendix

Lemma 9. If $(\Phi^j_{\mathtt{L}} \rhd_{\mathcal{A}} \Gamma_j \Vdash \mathtt{L} \rhd \Delta_j)_{j\in J}$, then $\Phi_{\mathtt{L}} \rhd_{\mathcal{A}} +_{j\in J}\Gamma_j \Vdash \mathtt{L} \rhd +_{j\in J}\Delta_j$. Moreover, $\mathtt{sz}(\Phi_{\mathtt{L}}) = +_{j\in J}\mathtt{sz}(\Phi^j_{\mathtt{L}}) - (\mathtt{height}(\mathtt{L}) \cdot (|J| - 1))$.

Proof. The proof is by induction on $\mathtt{L}$. The case $\mathtt{L} = \Box$ is straightforward since $\Gamma_j = \Delta_j = \emptyset$ so that $+_{j\in J}\Gamma_j = +_{j\in J}\Delta_j = \emptyset$. We have $\mathtt{sz}(\Phi_\Box) = 1 = |J| - |J| + 1 = |J| - [1 \cdot (|J| - 1)] = +_{j\in J}\mathtt{sz}(\Phi^j_\Box) - (\mathtt{height}(\Box) \cdot (|J| - 1))$.

Let $\mathtt{L} = \mathtt{L}'[y/u]$. Then for each $j \in J$, the derivation $\Phi^j_{\mathtt{L}}$ has the following form, where $\Gamma_j = \Pi_j +_{i\in K_j \uplus H_j} \Lambda^j_k$ and $\Delta_j = \Delta'_j; y : \{\!\{\sigma_h\}\!\}_{h\in H_j}$.

$$\frac{\Phi^j_{\mathtt{L}'} \rhd y : \{\!\{\sigma_k\}\!\}_{k\in K_j}; \Pi_j \Vdash \mathtt{L}' \rhd \Delta'_j \qquad (\Psi^{j,i}_u \rhd \Lambda^j_i \vdash u : \sigma_i)_{i\in K_j\uplus H_j}}{\Pi_j +_{i\in K_j\uplus H_j} \Lambda^j_i \Vdash \mathtt{L}'[y/u] \rhd \Delta'_j; y : \{\!\{\sigma_h\}\!\}_{h\in H_j}}$$

By the *i.h.* we have $\Phi_{\mathtt{L}'} \rhd +_{j\in J}(y : \{\!\{\sigma_k\}\!\}_{k\in K_j}; \Pi_j) \Vdash \mathtt{L}' \rhd +_{j\in J}\Delta'_j$, but $+_{j\in J}(y : \{\!\{\sigma_k\}\!\}_{k\in K_j}; \Pi_j) = y : \{\!\{\sigma_k\}\!\}_{j\in J, k\in K_j}; +_{j\in J}\Pi_j$. We thus construct the derivation $\Phi_{\mathtt{L}}$:

$$\frac{\Phi_{\mathtt{L}'} \rhd y : \{\!\{\sigma_k\}\!\}_{j\in J,k\in K_j}; +_{j\in J}\Pi_j \Vdash \mathtt{L}' \rhd +_{j\in J}\Delta'_j \qquad (\Psi^{j,i}_u \rhd \Lambda^j_i \vdash u : \sigma_i)_{j\in J, i\in K_j\uplus H_j}}{+_{j\in J}\Pi_j +_{j\in J, i\in K_j\uplus H_j} \Lambda^j_i \Vdash \mathtt{L}'[y/u] \rhd +_{j\in J}\Delta'_j; y : \{\!\{\sigma_h\}\!\}_{j\in J, h\in H_j}}$$

We conclude with the first statement since $+_{j\in J}\Gamma_j = +_{j\in J}\Pi_j +_{j\in J, i\in K_j\uplus H_j} \Lambda^j_k$ and $+_{j\in J}\Delta_j = +_{j\in J}\Delta'_j, y : \{\!\{\sigma_h\}\!\}_{j\in J, h\in H_j}$. Moreover:

$\mathtt{sz}(\Phi_{\mathtt{L}}) = \mathtt{sz}(\Phi_{\mathtt{L}'}) +_{j\in J, i\in K_j\uplus H_j} \mathtt{sz}(\Phi_{\Psi^{j,i}_u}) + 1 =_{i.h.}$
$+_{j\in J}\mathtt{sz}(\Phi_{\mathtt{L}'}) - (\mathtt{height}(\mathtt{L}') \cdot (|J| - 1)) +_{j\in J, i\in K_j\uplus H_j} \mathtt{sz}(\Phi_{\Psi^{j,i}_u}) + 1 =_{i.h.}$
$+_{j\in J}\mathtt{sz}(\Phi_{\mathtt{L}'}) +_{j\in J, i\in K_j\uplus H_j} \mathtt{sz}(\Phi_{\Psi^{j,i}_u}) + |J| - [(\mathtt{height}(\mathtt{L}') + 1) \cdot (|J| - 1)] =$
$+_{j\in J}\mathtt{sz}(\Phi_{\mathtt{L}}) - (\mathtt{height}(\mathtt{L}) \cdot (|J| - 1))$ ◇

Theorem 4 (Weighted Subject Reduction for need). Let $\Phi \rhd_{\mathcal{A}} \Gamma \vdash t{:}\tau$. If $t \to_{\mathtt{need}} t'$, then there exists Φ' s.t. $\Phi' \rhd_{\mathcal{A}} \Gamma \vdash t'{:}\tau$. Moreover, $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.

Proof. By induction on $t \to_{\mathtt{need}} t'$.

- If $t = \mathtt{L}[\lambda x.u]s \to_{\mathtt{dB}} \mathtt{L}[u[x/s]] = t'$, then one shows $\Phi' \rhd \Gamma \vdash t'{:}\tau$ and $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$ by induction on $\mathtt{L}$. See the first case of Lem. 2 in [23] (pp. 19).
- If $t = \mathtt{N}[\![x]\!][x/\mathtt{L}[u]] \to_{\mathtt{lsv}} \mathtt{L}[\mathtt{N}[\![u]\!][x/u]] = t'$, then the derivation Φ has the following form, where $\Gamma = \Gamma_0 +_{i\in I} \Delta_i$.

$$\frac{\Phi_{\mathtt{N}[\![x]\!]} \rhd x : \{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma_0 \vdash \mathtt{N}[\![x]\!] : \sigma \qquad \left(\Phi^i_{\mathtt{L}[u]} \rhd \Delta_i \vdash \mathtt{L}[u] : \sigma_i\right)_{i\in I}}{\Gamma_0 +_{i\in I} \Delta_i \vdash \mathtt{N}[\![x]\!][x/\mathtt{L}[u]] : \sigma}$$

  By Lem. 8, for all $i \in I$, there exist $\Pi^i_1, \Pi^i_2, \Pi^i_3$ such that $\Phi^i_{\mathtt{L}} \rhd \Pi^i_1 \Vdash \mathtt{L} \rhd \Pi^i_2$, $\Phi^i_u \rhd \Pi^i_2; \Pi^i_3 \vdash u : \sigma_i$ and $\Delta_i = \Pi^i_1 + \Pi^i_3$.
  Then, from the derivations $\Phi_{\mathtt{N}[\![x]\!]}$ and $(\Phi^i_u)_{i\in I}$ we get, by Lem. 10, a derivation $\Phi_{\mathtt{N}[\![u]\!]} \rhd x : \{\!\{\sigma_i\}\!\}_{i\in I\setminus K}; \Gamma_0 +_{k\in K} (\Pi^k_2; \Pi^k_3) \vdash \mathtt{N}[\![u]\!] : \sigma$ for some $K \subseteq I$. Therefore, we can construct the following derivation $\Phi_{\mathtt{N}[\![u]\!][x/u]}$.

$$\frac{\Phi_{\mathtt{N}[\![u]\!]} \qquad \left(\Phi^i_u\right)_{i\in I\setminus K}}{\Gamma_0 +_{k\in K} (\Pi^k_2;\Pi^k_3) +_{i\in I\setminus K} (\Pi^i_2;\Pi^i_3) \vdash \mathtt{N}[\![u]\!][x/u] : \sigma}$$

The last sequent can be written $\Gamma_0 + (+_{i\in I}\Pi^i_2; +_{i\in I}\Pi^i_3) \vdash \mathtt{N}[\![u]\!][x/u] : \sigma$. Now, from $\Phi_{\mathtt{N}[\![x]\!]}$ and Lem. 11 we know that $I \neq \emptyset$. We can thus apply Lem. 9 to $(\Phi^i_{\mathtt{L}})_{i\in I}$ and we get $\Phi_{\mathtt{L}} \triangleright +_{i\in I}\Pi^i_1 \Vdash \mathtt{L} \triangleright +_{i\in I}\Pi^i_2$. We can thus apply Lem. 8 to $\Phi_{\mathtt{L}}$ and $\Phi_{\mathtt{N}[\![u]\!][x/u]}$, obtaining $\Phi' \triangleright \Gamma_0 +_{i\in I} \Pi^1_1 +_{i\in I} \Pi^i_3 \vdash L[\mathtt{N}[\![u]\!][x/u]] : \sigma$. We can then conclude with the first statement since $\Gamma_0 +_{i\in I} \Pi^1_1 +_{i\in I} \Pi^i_3 = \Gamma_0 +_{i\in I} \Delta_i = \Gamma$ as required. Moreover, for the second one, we have two equalities:

$$\begin{array}{ll}
\mathtt{sz}(\Phi) = \mathtt{sz}(\Phi_{\mathtt{N}[\![x]\!]}) +_{i\in I} \mathtt{sz}(\Phi^i_{\mathtt{L}[u]}) + 1 & =_{L.\ 8} \\
\mathtt{sz}(\Phi_{\mathtt{N}[\![x]\!]}) +_{i\in I} (\mathtt{sz}(\Phi^i_{\mathtt{L}}) + \mathtt{sz}(\Phi^i_u) - 1) + 1 & = \\
\mathtt{sz}(\Phi_{\mathtt{N}[\![x]\!]}) +_{i\in I} \mathtt{sz}(\Phi^i_{\mathtt{L}}) +_{i\in I} \mathtt{sz}(\Phi^i_u) - (|I| - 1) = Z - (|I| - 1) &
\end{array}$$

and

$$\begin{array}{ll}
\mathtt{sz}(\Phi') =_{L.\ 8} \mathtt{sz}(\Phi_{\mathtt{L}}) + \mathtt{sz}(\Phi_{\mathtt{N}[\![u]\!][x/u]}) - 1 & = \\
\mathtt{sz}(\Phi_{\mathtt{L}}) + \mathtt{sz}(\Phi_{\mathtt{N}[\![u]\!]}) +_{i\in I\setminus K} \mathtt{sz}(\Phi^i_u) + 1 - 1 & = \\
\mathtt{sz}(\Phi_{\mathtt{L}}) + \mathtt{sz}(\Phi_{\mathtt{N}[\![u]\!]}) +_{i\in I\setminus K} \mathtt{sz}(\Phi^i_u) & =_{L.\ 10} \\
\mathtt{sz}(\Phi_{\mathtt{L}}) + \mathtt{sz}(\Phi_{\mathtt{N}[\![x]\!]}) +_{k\in K} \mathtt{sz}(\Phi^k_u) - |K| +_{i\in I\setminus K} \mathtt{sz}(\Phi^i_u) & =_{L.\ 9} \\
+_{i\in I}\mathtt{sz}(\Phi^i_{\mathtt{L}}) - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] + \mathtt{sz}(\Phi_{\mathtt{N}[\![x]\!]}) +_{i\in I} \mathtt{sz}(\Phi^i_u) - |K| = & \\
\mathtt{sz}(\Phi_{\mathtt{N}[\![x]\!]}) +_{i\in I} \mathtt{sz}(\Phi^i_{\mathtt{L}}) +_{i\in I} \mathtt{sz}(\Phi^i_u) - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] - |K| & = \\
Z - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] - |K| &
\end{array}$$

To conclude, we know by Lem. 10 that $K \neq \emptyset$. Therefore, $|I|-1 \leq \mathtt{height}(\mathtt{L}) \cdot (|I| - 1)$ so that $Z - (|I| - 1) \geq Z - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] > Z - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] - |K|$. We thus conclude $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$ as required.

– If $t = \mathtt{N}_1[\![x]\!][x/\mathtt{N}_2[u]] \to_{\mathtt{need}} \mathtt{N}_1[\![x]\!][x/\mathtt{N}_2[u']] = t'$ comes from $\mathtt{N}[u] \to_{\mathtt{need}} \mathtt{N}[u']$, then Φ has the following form, where $\Gamma = \Gamma_0 +_{i\in I} \Delta_i$.

$$\frac{x : \{\!\{\sigma_i\}\!\}_{i\in I}; \Gamma_0 \vdash \mathtt{N}_1[\![x]\!]{:}\tau \qquad \left(\Phi^i_{\mathtt{N}_2[u]} \triangleright \Delta_i \vdash \mathtt{N}_2[u]{:}\sigma_i\right)_{i\in I}}{\Gamma_0 +_{i\in I} \Delta_i \vdash \mathtt{N}_1[\![x]\!][x/\mathtt{N}_2[u]]{:}\tau}$$

By the *i.h.* both $\Pi^i_{\mathtt{N}_2[u]} \triangleright \Delta_i \vdash \mathtt{N}_2[u']{:}\sigma_i$ and $\mathtt{sz}(\Phi^i_{\mathtt{N}_2[u]}) > \mathtt{sz}(\Pi^i_{\mathtt{N}_2[u]})$ hold for all $i \in I$. The first statement gives $\Phi' \triangleright \Gamma_0 +_{i\in I} \Delta_i \vdash \mathtt{N}_1[\![x]\!][x/\mathtt{N}_2[u']]{:}\tau$. The second one gives $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$ since $I \neq \emptyset$ holds by Lem. 11.

– The cases $t = \mathtt{N}[u]v \to_{\mathtt{need}} \mathtt{N}[u']v = t'$ and $t = \mathtt{N}[u][x/v] \to_{\mathtt{need}} \mathtt{N}[u'][x/v] = t'$ coming from $\mathtt{N}[u] \to_{\mathtt{need}} \mathtt{N}[u']$ are straightforward inductive cases. ◇