Intersection Types as Evaluation Types

Pablo Barenbaum[1], Eduardo Bonelli[2], and Mariana Milicich[3‡§]

[1] Universidad Nacional de Quilmes (CONICET), and Instituto de Ciencias de la Computación, UBA, Argentina
[2] Stevens Institute of Technology, USA
[3] Université Paris Cité, CNRS, IRIF, France

This work explores the idea that intersection types may be seen as a standardised representation of reduction sequences to normal form within the context of different reduction strategies. We propose two non-idempotent intersection type systems: one for weak call-by-name and another one for weak and closed call-by-value. The key feature of both type systems is that the type of a term is given by its normal form. As a main result, we show there is a one-to-one correspondence between typing derivations and reduction sequences to normal form. Specifically, we demonstrate that each reduction to a normal form N corresponds to a typing derivation which has N as its type, and that the converse holds. This result refines the characterisation of normalisation via intersection types.

1 Introduction

The goal of this work is to show that typing derivations can be used as a *notation* for reduction sequences to normal form[1], by typing any normalising term with its normal form. Hence the name "evaluation type systems". Schematically speaking, we establish a one-to-one correspondence between reductions $t \to^* s$ and derivations $\vdash t : s$, where s is a normal form.

We assume the reader has some familiarity with *intersection types* (IT), which is the technical tool used in this work, as IT characterise (strong) normalisation [6]. Our starting point are *non-idempotent* IT [8, 5], where typing derivations and reduction sequences to normal form become more closely related in such type systems; *e.g.* lengths of normalisation sequences can be measured exactly [1]. Moreover, since we establish a one-to-one correspondence between typing derivations and normalisation sequences, our result refines the characterisation of normalisation mentioned above, and it is done in the context of weak call-by-name (CBN) and weak and closed call-by-value (CBV).

Several works have explored non-idempotent IT in the context of CBN [4, 2] and CBV [7, 9]. So far, we are unaware of existing intersection type systems explicitly typing a term with its normal form. However, Bernadet and Graham-Lengrand's type system for strong CBN in [3] types terms with the *structure*[2] of their corresponding normal form. Their work differs from ours in the sense that they focus on the quantitative information that can be derived from the type system. In our case, we want to refine the gap between typing and computation, using a type system as a notation for reduction in the context of two different reduction strategies.

‡ *This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 945332.*
§ Work by this author was partially supported by Instituto de Ciencias de la Computación, UBA, Argentina.

[1] Also named **normalisation sequences**.
[2] The *structure* of a term is given by its shape, forgetting all variable names.

2 Call-by-Name

In this section, we provide the common syntax we use in the document. Then we recall the operational semantics of CBN, and we propose an evaluation type system based on non-idempotent IT. The main result is a one-to-one correspondence between the set of typing derivations and the set of normalisation sequences (Theorem 2.3).

Given a denumerable set of **variables** $(x, y, z, \ldots)$, the sets of **terms** $(t, s, \ldots)$ and **answers** $(\mathtt{a}, \mathtt{b}, \ldots)$ are given by the grammar:

$$t, s ::= x \mid \lambda x.\, t \mid t\, s \qquad \mathtt{a} ::= \lambda x.\, t \mid x\, t_1 \ldots t_n$$

Intuitively, the set of answers is the set of normal forms. Capture-avoiding substitution of free occurrences of x in t by s is written $t\{x := s\}$. **CBN reduction** is given by the rules:

$$\frac{}{(\lambda x.\, t)\, s \to_{\mathsf{name}} t\{x := s\}}\,\beta \qquad \frac{t \to_{\mathsf{name}} t'}{t\, s \to_{\mathsf{name}} t'\, s}\,\mu$$

The sets of **types** $(A, B, \ldots)$ and **multitypes** $(M, N, \ldots)$ are given by the grammar:

$$\begin{array}{rcll} A & ::= & \mathtt{a} & \text{answer types} \\ & \mid & t.M \to A & \text{arrow types} \end{array} \qquad M ::= [A_1, \ldots, A_n] \qquad (n \geq 0)$$

where multitypes are multisets of types. Intuitively, an answer type $\mathtt{a}$ corresponds to the normal form of the term being typed. Meanwhile, an arrow type $t.M \to A$ can be associated with an abstraction $\lambda x.\, s$ such that the free occurrences of x in s shall become bound to t, where in turn t is assigned the multitype M.

If x is a variable and t is a term, we write $x : t$ to denote the **association** of x to t. **Substitutions** $(\sigma, \sigma', \ldots)$ are finite sets of associations of the form $(x_1 : t_1, \ldots, x_n : t_n)$, such that $x_i \neq x_j$ for all $i \neq j$. If $\sigma = (x_1 : t_1, \ldots, x_n : t_n)$, its **domain** is $\mathsf{dom}(\sigma) := \{x_1, \ldots, x_n\}$, and we write $\sigma(x)$ to denote the **substitution** of x by σ, defined by declaring that $\sigma(x_i) := t_i$ for all $i \in 1..n$ and $\sigma(y) = y$ for every other variable. The capture-avoiding substitution of each free occurrence of every variable $x \in \mathsf{dom}(\sigma)$ in t by $\sigma(x)$ is written t^σ, and defined recursively, taking as base case $x^\sigma := \sigma(x)$. A substitution σ is **idempotent** if $(t^\sigma)^\sigma = t^\sigma$ holds for any term t. A substitution σ is **strictly idempotent** if it is idempotent and $\sigma(x) \neq x$ holds for every variable $x \in \mathsf{dom}(\sigma)$. Henceforth, as a convention, when referring to σ, σ', etc., we assume that they stand for *strictly idempotent* substitutions. We write $\varnothing$ for the empty substitution, and $\sigma, x : t$ for the extension of σ with the association $x : t$.

Typing contexts $(\Gamma, \Gamma', \ldots)$, are functions mapping each variable to a multitype, in such a way that only finitely many variables are assigned a non-empty multitype. The **domain** of a typing context Γ is denoted $\mathsf{dom}(\Gamma)$ and is defined as the finite set of variables assigned to a non-empty multitype, expressed as $\mathsf{dom}(\Gamma) := \{x \mid \Gamma(x) \neq [\,]\}$. If M_1 and M_2 are multitypes, its **sum** $M_1 + M_2$ is defined as the additive union of multisets: $[A_1, \ldots, A_n] + [B_1, \ldots, B_m] := [A_1, \ldots, A_n, B_1, \ldots, B_m]$. If Γ_1, Γ_2 are typing contexts, its **sum** $\Gamma_1 + \Gamma_2$ is defined as the point-wise sum: $(\Gamma_1 + \Gamma_2)(x) := \Gamma_1(x) + \Gamma_2(x)$. We write $\Gamma +_{i=1}^{n} \Delta_i$ for the sum $\Gamma + \Delta_1 + \ldots + \Delta_n$, and similarly for multitypes. We write $x : M$ for the typing context such that $\mathsf{dom}(x : M) = \{x\}$ and $(x : M)(x) = M$. Sometimes we write Γ, Δ for the **disjoint sum of typing contexts**, when $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \varnothing$. Since the domain of a typing context is finite, we can always write a typing context as of the form $x_1 : M_1, \ldots, x_n : M_n$ using the grammar $\Gamma ::= \varnothing \mid \Gamma, x : M$.

Typing judgements are of the form $\sigma \,;\, \Gamma \vdash t \Downarrow_{\mathsf{name}} A$, where we recall that σ is always assumed to be a strictly idempotent substitution. Derivability of typing judgements is defined inductively by the following rules:

$$\frac{x \notin \mathsf{dom}(\sigma)}{\sigma \,;\, \varnothing \vdash x \Downarrow_{\mathsf{name}} x}\,\mathtt{n\text{-}var}_1 \qquad \frac{x \in \mathsf{dom}(\sigma) \qquad \mathsf{fv}(A) \,\#\, \mathsf{dom}(\sigma)}{\sigma \,;\, x : [A] \vdash x \Downarrow_{\mathsf{name}} A}\,\mathtt{n\text{-}var}_2$$

$$\frac{}{\sigma\,;\varnothing \vdash \lambda x.\,t \Downarrow_{\mathsf{name}} \lambda x.\,t^{\sigma}}\,\texttt{n-lam}_1 \qquad \frac{\sigma, x : s\,;\Gamma, x : M \vdash t \Downarrow_{\mathsf{name}} A}{\sigma\,;\Gamma \vdash \lambda x.\,t \Downarrow_{\mathsf{name}} s.M \to A}\,\texttt{n-lam}_2$$

$$\frac{\sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{name}} x\,t_1 \dots t_n}{\sigma\,;\Gamma \vdash t\,s \Downarrow_{\mathsf{name}} x\,t_1 \dots t_n\,s^{\sigma}}\,\texttt{n-app}_1$$

$$\frac{\sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{name}} s^{\sigma}.[A_1, \dots, A_n] \to B \qquad (\sigma\,;\Delta_i \vdash s \Downarrow_{\mathsf{name}} A_i)_{i=1}^{n}}{\sigma\,;\Gamma +_{i=1}^{n} \Delta_i \vdash t\,s \Downarrow_{\mathsf{name}} B}\,\texttt{n-app}_2$$

A **typing derivation** is a tree obtained from the rules above. We write $\mathsf{D} \rhd \sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{name}} A$ if D is a typing derivation concluding with the judgement $\sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{name}} A$.

There are two rules for typing each term in the syntax, depending on its role in the computation. A variable x may be typed using either $\texttt{n-var}_1$ or $\texttt{n-var}_2$, depending on whether it will be substituted by another term or not. For instance, consider the term $t := (\lambda x.\,x)\,y$, which reduces in one β-step to y, substituting x by y. We type t as follows:

$$\dfrac{\dfrac{\dfrac{x \in \{x\} \quad \{y\} \# \{x\}}{x : y\,;x : [y] \vdash x \Downarrow_{\mathsf{name}} y}\,\texttt{n-var}_2}{\varnothing\,;\varnothing \vdash \lambda x.\,x \Downarrow_{\mathsf{name}} y.[y] \to y}\,\texttt{n-lam}_2 \qquad \dfrac{y \notin \mathsf{dom}(\varnothing)}{\varnothing\,;\varnothing \vdash y \Downarrow_{\mathsf{name}} y}\,\texttt{n-var}_1}{\varnothing\,;\varnothing \vdash (\lambda x.\,x)\,y \Downarrow_{\mathsf{name}} y}\,\texttt{n-app}_2$$

The variable y is typed with the rule $\texttt{n-var}_1$ as it is not substituted during the computation of t. On the contrary, x is substituted in the body of $\lambda x.\,x$ by y, and it is typed with rule $\texttt{n-var}_2$. Furthermore, in the premise of rule $\texttt{n-lam}_2$, the empty substitution is expanded with the association $x : y$. An abstraction can be typed with rule $\texttt{n-lam}_1$ if it will not be applied to an argument during the computation of the top-level term. Otherwise, it is typed using rule $\texttt{n-lam}_2$, as seen in the typing derivation of t. For applications, there are two possibilities as well: either the normal form of t in $t\,s$ is of the form $x\,t_1\,\dots\,t_n$, in which case it is typed with rule $\texttt{n-app}_1$, or t is a redex, *i.e.*, of the form $(\lambda y.\,t')\,s_1\,\dots\,s_m$, in which case it is typed with rule $\texttt{n-app}_2$.

In what follows, we mention two properties the type system enjoys: Weighted Subject Reduction and Subject Expansion.

In order to state the Weighted Subject Reduction lemma, the notion of weight of a derivation must be considered. Given a derivation $\mathsf{D} \rhd \sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{name}} A$, the **weight** of D is written $W(\mathsf{D})$ and corresponds to the number of typing rules used in D.

Lemma 2.1 (Weighted Subject Reduction)**.** *If $t \to_{\mathsf{name}} t'$ and $\mathsf{D} \rhd \sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{name}} A$, then there exists a derivation $\mathsf{D}' \rhd \sigma\,;\Gamma \vdash t' \Downarrow_{\mathsf{name}} A$ such that $W(\mathsf{D}) > W(\mathsf{D}')$.*

Lemma 2.2 (Subject Expansion)**.** *If $t \to_{\mathsf{name}} t'$ and $\mathsf{D}' \rhd \sigma\,;\Gamma \vdash t' \Downarrow_{\mathsf{name}} A$, then there exists a derivation D such that $\mathsf{D} \rhd \sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{name}} A$.*

The proofs of both lemmas are by induction on the reduction step, employing techniques known by the community using the lemmas of Weighted Substitution and Anti-Substitution.

We define **CBN evaluation contexts** N with the following grammar: $\mathsf{N} ::= \Box \mid \mathsf{N}\,t$, and we write $\mathsf{N}\langle t\rangle$ for the substitution of $\Box$ in N by t. Then, **CBN normalisation sequences** $(\mathsf{R}, \mathsf{R}', \dots)$ are defined by: $\mathsf{R} ::= \epsilon_{\mathsf{a}} \mid \mathsf{N}\langle(\underline{\lambda}x.t)\,s\rangle\,;\mathsf{R}$.

CBN normalisation judgements are written $\mathsf{R} \rhd t \twoheadrightarrow_{\mathsf{name}} \mathsf{a}$, and are defined as:

$$\frac{}{\epsilon_{\mathsf{a}} \rhd \mathsf{a} \twoheadrightarrow_{\mathsf{name}} \mathsf{a}} \qquad \frac{\mathsf{R} \rhd \mathsf{N}\langle t\{x := s\}\rangle \twoheadrightarrow_{\mathsf{name}} \mathsf{a}}{(\mathsf{N}\langle(\underline{\lambda}x.t)\,s\rangle\,;\mathsf{R}) \rhd \mathsf{N}\langle(\lambda x.\,t)\,s\rangle \twoheadrightarrow_{\mathsf{name}} \mathsf{a}}$$

For instance, we have that $((\underline{\lambda}x.\lambda y.\,x)\,z\,w\,;(\underline{\lambda}y.z)\,w\,;\epsilon_z) \rhd (\lambda x.\,\lambda y.\,x)\,z\,w \twoheadrightarrow_{\mathsf{name}} z$.

The sets of **CBN derivations** $\mathcal{D}_{\mathsf{name}}$ and **CBN reductions to normal form** $\mathcal{R}_{\mathsf{name}}$ are defined by:

$$\mathcal{D}_{\mathsf{name}} := \{\mathsf{D} \mid \exists t, \mathtt{a}.\ \mathsf{D} \rhd \varnothing \,;\, \varnothing \vdash t \Downarrow_{\mathsf{name}} \mathtt{a}\} \qquad \mathcal{R}_{\mathsf{name}} := \{\mathsf{R} \mid \exists t, \mathtt{a}.\ \mathsf{R} \rhd t \twoheadrightarrow_{\mathsf{name}} \mathtt{a}\}$$

We now state the main result of this section, which establishes a one-to-one correspondence between the sets of typing derivations and normalisation sequences:

Theorem 2.3 (Correspondence between typing derivations and normalisation sequences)**.** *There exist mappings* $f : \mathcal{D}_{\mathsf{name}} \to \mathcal{R}_{\mathsf{name}}$ *and* $g : \mathcal{R}_{\mathsf{name}} \to \mathcal{D}_{\mathsf{name}}$ *such that:*

1. *If* $\mathsf{D} \rhd \varnothing \,;\, \varnothing \vdash t \Downarrow_{\mathsf{name}} \mathtt{a}$ *then* $f(\mathsf{D}) \rhd t \twoheadrightarrow_{\mathsf{name}} \mathtt{a}$.
2. *If* $\mathsf{R} \rhd t \twoheadrightarrow_{\mathsf{name}} \mathtt{a}$ *then* $g(\mathsf{R}) \rhd \varnothing \,;\, \varnothing \vdash t \Downarrow_{\mathsf{name}} \mathtt{a}$.

Furthermore, f *and* g *are mutual inverses.*

Proof. The proof relies on Weighted Subject Reduction and Subject Expansion. Moreover, we also need to prove the following auxiliary result:

Lemma 2.4 (Typing answers)**.** *Let* $\mathsf{D} \rhd \varnothing \,;\, \varnothing \vdash \mathtt{a} \Downarrow_{\mathsf{name}} A$*. Then:*

1. *If* $\mathtt{a}$ *is an abstraction* **and** A *is an answer,* **then** $A = \mathtt{a}$.
2. *If* $\mathtt{a}$ *is not an abstraction,* **then** A *is an answer* **and** $A = \mathtt{a}$.

Furthermore, in both cases, D *uses only instances of rules* $\mathtt{n\text{-}var}_1$*,* $\mathtt{n\text{-}lam}_1$*, and* $\mathtt{n\text{-}app}_1$*.* □

3 Closed Call-by-Value

In this section, we present the operational semantics of closed CBV. Then, we propose a type system based on non-idempotent IT. The main result is a one-to-one correspondence between the set of typing derivations and the set of normalisation sequences (Theorem 3.3).

We start by defining the set of **values** $(\mathtt{v}, \mathtt{w}, \ldots)$ with the grammar $\mathtt{v} ::= \lambda x.\, t$. The following rules inductively define **CBV reduction**:

$$\frac{}{(\lambda x.\, t)\,\mathtt{v} \to_{\mathsf{value}} t\{x := \mathtt{v}\}}\ \beta_v \qquad \frac{t \to_{\mathsf{value}} t'}{t\,s \to_{\mathsf{value}} t'\,s}\ \mu \qquad \frac{t \to_{\mathsf{value}} t'}{\mathtt{v}\,t \to_{\mathsf{value}} \mathtt{v}\,t'}\ \nu$$

The sets of **types** $(A, B, \ldots)$ and **multitypes** $(M, N, \ldots)$ are given by the grammar:

$$\begin{array}{rcll} A & ::= & \mathtt{v}.M \to \mathtt{w}.N & \\ M & ::= & [A_1, \ldots, A_n] & (n \geq 0) \end{array}$$

A type of the form $\mathtt{v}.M \to \mathtt{w}.N$ corresponds to the type of a function that is applied to a value $\mathtt{v}$, and return a value $\mathtt{w}$ as its result, where $\mathtt{v}$ and $\mathtt{w}$ have multitypes M and N respectively. The cardinality of a multitype corresponds to the number of times the term is *used* within its evaluation context. Specifically, a term is **used** in our closed CBV reduction if it is applied to a value. For instance, consider $s := (\lambda x.\, x)\,\lambda x.\, x$. In this example, the identity function on the left is used once, as it is applied to a value, whereas the one on the right is not used at all. We shall return to this later.

Regarding substitutions $(\sigma, \sigma', \ldots)$, they follow a definition akin to those in CBN, but CBV substitutions bind variables to values. Specifically, **substitutions** are finite sets of the form $(x_1 : \mathtt{v}_1, \ldots, x_n : \mathtt{v}_n)$, and we assume them to be strictly idempotent. On the other hand, **typing contexts** are defined as those for CBN: functions with finite domain, mapping each variable to a multitype.

Typing judgements can be of the form $\sigma\,;\Gamma \vdash \mathtt{v} \Downarrow_{\mathsf{value}} A$ and $\sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{value}} \mathtt{v}.M$. Derivability of typing judgements is defined inductively by the following rules:

$$\frac{x \in \mathsf{dom}(\sigma)}{\sigma\,; x : M \vdash x \Downarrow_{\mathsf{value}} \sigma(x).M}\ \mathtt{v\text{-}var} \qquad \frac{\sigma, x : \mathtt{v}\,;\Gamma, x : M \vdash t \Downarrow_{\mathsf{value}} \mathtt{w}.N}{\sigma\,;\Gamma \vdash \lambda x.\, t \Downarrow_{\mathsf{value}} \mathtt{v}.M \to \mathtt{w}.N}\ \mathtt{v\text{-}lam}$$

$$\frac{\sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{value}} \mathtt{v}.[\mathtt{w}.M \to \mathtt{u}.N] \qquad \sigma\,;\Delta \vdash s \Downarrow_{\mathsf{value}} \mathtt{w}.M}{\sigma\,;\Gamma + \Delta \vdash t\, s \Downarrow_{\mathsf{value}} \mathtt{u}.N}\ \mathtt{v\text{-}app}$$

$$\frac{(\sigma\,;\Gamma_i \vdash \lambda x.\, t \Downarrow_{\mathsf{value}} A_i)_{i=1}^{n}}{\sigma\,;+_{i=1}^{n}\Gamma_i \vdash \lambda x.\, t \Downarrow_{\mathsf{value}} (\lambda x.\, t^{\sigma}).[A_1, \ldots, A_n]}\ \mathtt{v\text{-}many}$$

We define **(typing) derivations** as in CBN. Note that abstractions can be typed in two ways: either with multitypes, as per rule `v-many`, or with types using rule `v-lam`. When an abstraction is *consuming*, *i.e.* when one of its descendants is the head of a β_v-redex, it is associated with a multitype whose cardinality is greater than 0, indicating there is at least one premise, that in turn is derived by rule `v-lam`. Conversely, if an abstraction is persistent, it is typed with the empty multitype. To illustrate this, we take the term t defined above, which reduces in one β_v-step to $\lambda x.\, x$. Abbreviating the second identity as `id`, and `v-many` as `v-m`, we type s as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{x \in \{x\}}{x : \mathtt{id}\,; x : [\,] \vdash x \Downarrow_{\mathsf{value}} \mathtt{id}.[\,]}\ \mathtt{v\text{-}var}}{\varnothing\,;\varnothing \vdash \lambda x.\, x \Downarrow_{\mathsf{value}} \mathtt{id}.[\,] \to \mathtt{id}.[\,]}\ \mathtt{v\text{-}lam}}{\varnothing\,;\varnothing \vdash \lambda x.\, x \Downarrow_{\mathsf{value}} \mathtt{id}.[\mathtt{id}.[\,] \to \mathtt{id}.[\,]]}\ \mathtt{v\text{-}m} \qquad \dfrac{}{\varnothing\,;\varnothing \vdash \mathtt{id} \Downarrow_{\mathsf{value}} \mathtt{id}.[\,]}\ \mathtt{v\text{-}m}}{\varnothing\,;\varnothing \vdash (\lambda x.\, x)\,\mathtt{id} \Downarrow_{\mathsf{value}} \mathtt{id}.[\,]}\ \mathtt{v\text{-}app}$$

This type system enjoys the properties of Weighted Subject Reduction and Subject Expansion:

Lemma 3.1 (Weighted Subject Reduction)**.** *Let $t \to_{\mathsf{value}} t'$ and let D be a derivation. If $\mathsf{D} \rhd \sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{value}} \mathtt{v}.M$ then there exists a derivation $\mathsf{D}' \rhd \sigma\,;\Gamma \vdash t' \Downarrow_{\mathsf{value}} \mathtt{v}.M$ such that $W(\mathsf{D}) > W(\mathsf{D}')$.*

Lemma 3.2 (Subject Expansion)**.** *Let $t \to_{\mathsf{value}} t'$. If $\sigma\,;\Gamma \vdash t' \Downarrow_{\mathsf{value}} \mathtt{v}.M$, then $\sigma\,;\Gamma \vdash t \Downarrow_{\mathsf{value}} \mathtt{v}.M$.*

We define **CBV evaluation contexts** V with the following grammar: $\mathsf{V} ::= \Box \mid \mathsf{V}\,t \mid \mathtt{v}\,\mathsf{V}$, and we write $\mathsf{V}\langle t\rangle$ for the substitution of $\Box$ in V by t. **CBV normalisation sequences** $(\mathsf{R}, \mathsf{R}', \ldots)$ are defined by: $\mathsf{R} ::= \epsilon_{\mathtt{v}} \mid \mathsf{V}\langle(\underline{\lambda}x.t)\,\mathtt{v}\rangle\ ;\ \mathsf{R}$. **CBV normalisation judgements** are written $\mathsf{R} \rhd t \twoheadrightarrow_{\mathsf{value}} \mathtt{v}$, and are defined as follows:

$$\frac{}{\epsilon_{\mathtt{v}} \rhd \mathtt{v} \twoheadrightarrow_{\mathsf{value}} \mathtt{v}} \qquad \frac{\mathsf{R} \rhd \mathsf{V}\langle t\{x := \mathtt{v}\}\rangle \twoheadrightarrow_{\mathsf{value}} \mathtt{w}}{(\mathsf{V}\langle(\underline{\lambda}x.t)\,\mathtt{v}\rangle\ ;\ \mathsf{R}) \rhd \mathsf{V}\langle(\lambda x.\, t)\,\mathtt{v}\rangle \twoheadrightarrow_{\mathsf{value}} \mathtt{w}}$$

The sets of **CBV derivations** $\mathcal{D}_{\mathsf{value}}$ and **CBV reductions to normal form** $\mathcal{R}_{\mathsf{value}}$ are defined by:

$$\mathcal{D}_{\mathsf{value}} := \{\mathsf{D} \mid \exists t, \mathtt{v}.\ \mathsf{D} \rhd \varnothing\,;\varnothing \vdash t \Downarrow_{\mathsf{value}} \mathtt{v}.[\,]\} \qquad \mathcal{R}_{\mathsf{value}} := \{\mathsf{R} \mid \exists t, \mathtt{v}.\ \mathsf{R} \rhd t \twoheadrightarrow_{\mathsf{value}} \mathtt{v}\}$$

Now we can establish the main result of this section:

Theorem 3.3 (Correspondence between typing derivations and reduction sequences)**.** *There exist mappings $f : \mathcal{D}_{\mathsf{value}} \to \mathcal{R}_{\mathsf{value}}$ and $g : \mathcal{R}_{\mathsf{value}} \to \mathcal{D}_{\mathsf{value}}$ such that:*

1. *If $\mathsf{D} \rhd \varnothing\,;\varnothing \vdash t \Downarrow_{\mathsf{value}} \mathtt{v}.[\,]$ then $f(\mathsf{D}) \rhd t \twoheadrightarrow_{\mathsf{value}} \mathtt{v}$.*
2. *If $\mathsf{R} \rhd t \twoheadrightarrow_{\mathsf{value}} \mathtt{v}$ then $g(\mathsf{R}) \rhd \varnothing\,;\varnothing \vdash t \Downarrow_{\mathsf{value}} \mathtt{v}.[\,]$.*

Furthermore, f and g are mutual inverses.

References

[1] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. *J. Funct. Program.*, 30:e14, 2020.

[2] 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*, volume 11423 of *Lecture Notes in Computer Science*, pages 410–439. Springer, 2019.

[3] Alexis Bernadet and Stéphane Graham-Lengrand. A big-step operational semantics via non-idempotent intersection types, 2013.

[4] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Log. J. IGPL*, 25(4):431–464, 2017.

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

[6] 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.

[7] 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.

[8] Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software*, pages 555–574. Springer, 1994.

[9] 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.