Towards a Semantic Measure of the Execution Time in Call-by-Value lambda-Calculus

Giulio Guerrieri
University of Bath, Department of Computer Science, Bath, United Kingdom
g.guerrieri@bath.ac.uk

We investigate the possibility of a semantic account of the execution time (i.e. the number of β_v-steps leading to the normal form, if any) for the shuffling calculus, an extension of Plotkin's call-by-value λ-calculus. For this purpose, we use a linear logic based denotational model that can be seen as a non-idempotent intersection type system: relational semantics. Our investigation is inspired by similar ones for linear logic proof-nets and untyped call-by-name λ-calculus. We first prove a qualitative result: a (possibly open) term is normalizable for weak reduction (which does not reduce under abstractions) if and only if its interpretation is not empty. We then show that the size of type derivations can be used to measure the execution time. Finally, we show that, differently from the case of linear logic and call-by-name λ-calculus, the quantitative information enclosed in type derivations does not lift to types (i.e. to the interpretation of terms). To get a truly semantic measure of execution time in a call-by-value setting, we conjecture that a refinement of its syntax and operational semantics is needed.

1 Introduction

Type systems enforce properties of programs, such as termination or deadlock-freedom. The guarantee provided by most type systems for the λ-calculus is *termination*.

Intersection types have been introduced as a way of extending simple types for the λ-calculus to "finite polymorphism", by adding a new type constructor $\cap$ and new typing rules governing it. Contrary to simple types, intersection types provide a sound and *complete* characterization of termination: not only typed programs terminate, but all terminating programs are typable as well (see [19, 20, 42, 36] where different intersection type systems characterize different notions of normalization). Intersection types are idempotent, that is, they verify the equation $A \cap A = A$. This corresponds to an interpretation of a typed term $t : A \cap B$ as "t can be used both as data of type A and as data of type B".

More recently [24, 35, 38, 15, 16] (a survey can be found in [13]), *non-idempotent* variants of intersection types have been introduced: they are obtained by dropping the equation $A \cap A = A$. In a non-idempotent setting, the meaning of the typed term $t : A \cap A \cap B$ is refined as "t can be used twice as data of type A and once as data of type B". This could give to programmers a way to keep control on the performance of their code and to count resource consumption. Finite multisets are the natural setting to interpret the associative, commutative and non-idempotent connective $\cap$: if A and B are non-idempotent intersection types, the multiset $[A, A, B]$ represents the non-idempotent intersection type $A \cap A \cap B$.

Non-idempotent intersection types have two main features, both enlightened by de Carvalho [15, 16]:

1. *Bounds on the execution time*: they go beyond simply qualitative characterisations of termination, as type derivations provide *quantitative* bounds on the execution time (*i.e.* on the number of β-steps to reach the β-normal form). Therefore, non-idempotent intersection types give intensional insights on programs, and seem to provide a tool to reason about complexity of programs. The approach is defining a measure for type derivations and showing that the measure gives (a bound to) the length of the evaluation of typed terms.

© G. Guerrieri
This work is licensed under the
Creative Commons Attribution License.

2. *Linear logic interpretation*: non-idempotent intersection types are deeply linked to linear logic (LL) [25]. Relational semantics [26, 11] — the category **Rel** of sets and relations endowed with the comonad ! of finite multisets — is a sort of "canonical" denotational model of LL; the Kleisli category $\mathbf{Rel}_!$ of the comonad ! is a CCC and then provides a denotational model of the ordinary (*i.e.* call-by-name) λ-calculus. Non-idempotent intersection types can be seen as a syntactic presentation of $\mathbf{Rel}_!$: the semantics of a term t is the set of conclusions of all type derivations of t.

These two facts together have a potential, fascinating consequence: denotational semantics may provide abstract tools for complexity analysis, that are theoretically solid, being grounded on LL.

Starting from [15, 16], research on relational semantics/non-idempotent intersection types has proliferated: various works in the literature explore their power in bounding the execution time or in characterizing normalization [17, 12, 10, 33, 9, 18, 40, 34, 13, 37, 3]. All these works study relational semantics/non-idempotent intersection types either in LL proof-nets (the graphical representation of proofs in LL), or in some variant of ordinary (*i.e.* call-by-name) λ-calculus. In the second case, the construction of the relational model $\mathbf{Rel}_!$ sketched above essentially relies on Girard's call-by-name translation $(\cdot)^n$ of intuitionistic logic into LL, which decomposes the intuitionistic arrow as $(A \Rightarrow B)^n = {!A^n} \multimap B^n$.

Ehrhard [22] showed that the relational semantics **Rel** of LL induces also a denotational model for the *call-by-value* λ-calculus[1] that can still be viewed as a non-idempotent intersection type system. The syntactic counterpart of this construction is Girard's ("boring") call-by-value translation $(\cdot)^v$ of intuitionistic logic into LL [25], which decomposes the intuitionistic arrow as $(A \Rightarrow B)^v = {!(A^v \multimap B^v)}$. Just few works have started the study of relational semantics/non-idempotent intersection types in a call-by-value setting [22, 21, 14, 23], and no one investigates their bounding power on the execution time in such a framework. Our paper aims to fill this gap and study the information enclosed in relational semantics/non-idempotent intersection types concerning the execution time in the call-by-value λ-calculus.

A difficulty arises immediately in the qualitative characterization of call-by-value normalization via the relational model. One would expect that the semantics of a term t is non-empty if and only if t is (strongly) normalizable for (some restriction of) the call-by-value evaluation $\to_{\beta_v}$, but it is impossible to get this result in Plotkin's original call-by-value λ-calculus λ_v [41]. Indeed, the terms t and u below are β_v-normal but their semantics in the relational model are empty:

$$t := (\lambda y.\Delta)(zI)\Delta \qquad u := \Delta((\lambda y.\Delta)(zI)) \qquad (\text{where } \Delta := \lambda x.xx \text{ and } I := \lambda x.x) \tag{1}$$

Actually, t and u should behave like the famous divergent term $\Delta\Delta$, since in λ_v they are observationally equivalent to $\Delta\Delta$ with respect all closing contexts and have the same semantics as $\Delta\Delta$ in all non-trivial denotational models of Plotkin's λ_v.

The reason of this mismatching is that in λ_v there are *stuck β-redexes* such as $(\lambda y.\Delta)(zI)$ in Eq. (1), *i.e.* β-redexes that β_v-reduction will never fire because their argument is normal but not a value (nor will it ever become one). The real problem with stuck β-redexes is that they may prevent the creation of other β_v-redexes, providing *"premature" β_v-normal forms* like t and u in Eq. (1). The issue affects termination and thus can impact on the study of observational equivalence and other operational properties in λ_v.

In a call-by-value setting, the issue of stuck β-redexes and then of premature β_v-normal forms arises only with *open terms* (in particular, when the reduction under abstractions is allowed, since it forces to deal with "locally open" terms). Even if to model functional programming languages with a call-by-value parameter passing, such as OCaml, it is usually enough to just consider closed terms and weak evaluation

[1]In call-by-value evaluation $\to_{\beta_v}$, function's arguments are evaluated before being passed to the function, so that β-redexes can fire only when their arguments are values, *i.e.* abstractions or variables. The idea is that only values can be erased or duplicated. Call-by-value evaluation is the most common parameter passing mechanism used by programming languages.

(*i.e.* not reducing under abstractions: function bodies are evaluated only when all parameters are supplied), the importance to consider open terms in a call-by-value setting can be found, for example, in partial evaluation (which evaluates a function when not all parameters are supplied, see [32]), in the theory of proof assistants such as Coq (in particular, for type checking in a system based on dependent types, see [27]), or to reason about (denotational or operational) equivalences of terms in λ_v that are congruences, or about other theoretical properties of λ_v such as separability or solvability [39, 45, 7, 14].

To overcome the issue of stuck β-redexes, we study relational semantics/non-idempotent intersection types in the *shuffling calculus* λ_{sh}, a conservative extension of Plotkin's λ_v proposed in [14] and further studied in [28, 30, 4, 31]. It keeps the same term syntax as λ_v and adds to β_v-reduction two commutation rules, σ_1 and σ_3, which "shuffle" constructors in order to move stuck β-redexes: they unblock β_v-redexes that are hidden by the "hyper-sequential structure" of terms. These commutation rules (referred also as *σ-reduction rules*) are similar to Regnier's σ-rules for the call-by-name λ-calculus [43, 44] and are inspired by the aforementioned $(\cdot)^v$ translation of the λ-calculus into LL proof-nets.

Following the same approach used in [16] for the call-by-name λ-calculus and in [17] for LL proof-nets, we prove that in the shuffling calculus λ_{sh}:

1. (*qualitative result*) relational semantics is adequate for λ_{sh}, *i.e.* a possibly open term is normalizable for weak reduction (not reducing under λ's) if and only if its interpretation in relational semantics is not empty (Thm. 16); this result was already proven in [14] using different techniques;
2. (*quantiative result*) the size of type derivations can be used to measure the execution time, *i.e.* the number of β_v-steps (and not σ-steps) to reach the normal form of the weak reduction (Prop. 21).

Finally, we show that, differently from the case of LL and call-by-name λ-calculus, we are *not* able to lift the quantitative information enclosed in type derivations to types (*i.e.* to the interpretation of terms) following the same technique used in [16, 17], as our Ex. 28 shows. In order to get a genuine semantic measure of execution time in a call-by-value setting, we conjecture that a refinement of its syntax and operational semantics is needed.

Even if our main goal has not yet been achieved, this investigation led to new interesting results:

1. all normalizing weak reduction sequences (if any) in λ_{sh} from a given term have the same number of β_v-steps (Cor. 22); this is not obvious, as we shall explain in Ex. 23;
2. terms whose weak reduction in λ_{sh} ends in a value has an elegant semantic characterization (Prop. 18), and the number of β_v-steps needed to reach their normal form can be computed in a simple way from a specific type derivation (Thm. 24).
3. all our qualitative and quantitative results for λ_{sh} are still valid in Plotkin's λ_v restricted to closed terms (which models functional programming languages), see Thm. 25, Cor. 26 and Thm. 27.

Proofs are omitted. They can be found in [29], the extended version of this paper.

1.1 Preliminaries and notations

The set of λ-terms is denoted by Λ. We set $I := \lambda x.x$ and $\Delta := \lambda x.xx$. Let $\to_{\mathsf{r}} \subseteq \Lambda \times \Lambda$.

- The reflexive-transitive closure of $\to_{\mathsf{r}}$ is denoted by $\to_{\mathsf{r}}^*$. The r-*equivalence* $\simeq_{\mathsf{r}}$ is the reflexive-transitive and symmetric closure of $\to_{\mathsf{r}}$.
- Let t be a term: t is r-*normal* if there is no term u such that $t \to_{\mathsf{r}} u$; t is r-*normalizable* if there is a r-normal term u such that $t \to_{\mathsf{r}}^* u$, and we then say that u is a r-*normal form of* t; t is *strongly* r-*normalizable* if there is no infinite sequence $(t_i)_{i\in\mathbb{N}}$ of terms such that $t = t_0$ and $t_i \to_{\mathsf{r}} t_{i+1}$ for all $i \in \mathbb{N}$. Finally, $\to_{\mathsf{r}}$ is *strongly normalizing* if every $u \in \Lambda$ is strongly r-normalizable.

$$\begin{array}{rll}
\textit{terms}: & t,u,s ::= v \mid tu & \text{(set: } \Lambda) \\
\textit{values}: & v ::= x \mid \lambda x.t & \text{(set: } \Lambda_v) \\
\textit{contexts}: & C ::= \langle\cdot\rangle \mid \lambda x.C \mid Ct \mid tC & \text{(set: } \Lambda_C) \\
\textit{Balanced contexts}: & B ::= \langle\cdot\rangle \mid (\lambda x.B)t \mid Bt \mid tB & \text{(set: } \Lambda_B)
\end{array}$$

$$\textit{Root-steps}: \quad (\lambda x.t)v \mapsto_{\beta_v} t\{v/x\} \qquad (\lambda x.t)us \mapsto_{\sigma_1} (\lambda x.ts)u,\ x \notin \mathsf{fv}(s) \qquad v((\lambda x.s)u) \mapsto_{\sigma_3} (\lambda x.vs)u,\ x \notin \mathsf{fv}(v)$$

$$\mapsto_{\sigma} := \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3} \qquad \mapsto_{\mathsf{sh}} := \mapsto_{\beta_v} \cup \mapsto_{\sigma}$$

$$\mathsf{r}\textit{-reduction}: \quad t \to_{\mathsf{r}} u \iff \exists C \in \Lambda_C, \exists t',u' \in \Lambda : t = C\langle t'\rangle, u = C\langle u'\rangle, t' \mapsto_{\mathsf{r}} u' \qquad \mathsf{r} \in \{\beta_v, \sigma_1, \sigma_3, \sigma, \mathsf{sh}\}$$

$$\mathsf{r}^\flat\textit{-reduction}: \quad t \to_{\mathsf{r}^\flat} u \iff \exists B \in \Lambda_B, \exists t',u' \in \Lambda : t = B\langle t'\rangle, u = B\langle u'\rangle, t' \mapsto_{\mathsf{r}} u' \qquad \mathsf{r} \in \{\beta_v, \sigma_1, \sigma_3, \sigma, \mathsf{sh}\}$$

Figure 1: The shuffling λ-calculus λ_{sh} [14].

- $\to_{\mathsf{r}}$ is *confluent* if ${}_{\mathsf{r}}^{*}\!\leftarrow \cdot \to_{\mathsf{r}}^{*} \subseteq \to_{\mathsf{r}}^{*} \cdot {}_{\mathsf{r}}^{*}\!\leftarrow$. From confluence it follows that: $t \simeq_{\mathsf{r}} u$ iff $t \to_{\mathsf{r}}^{*} s \; {}_{\mathsf{r}}^{*}\!\leftarrow u$ for some term s; and any r-normalizable term has a *unique* r-normal form.

2 The shuffling calculus

In this section we introduce the *shuffling calculus* λ_{sh}, namely the call-by-value λ-calculus defined in [14] and further studied in [28, 30, 4, 31]: it adds two commutation rules — the σ_1- and σ_3-reductions — to Plotkin's pure (i.e. without constants) call-by-value λ-calculus λ_v [41]. The syntax for terms of λ_{sh} is the same as Plotkin's λ_v and then the same as the ordinary (i.e. call-by-name) λ-calculus, see Fig. 1.

Clearly, $\Lambda_v \subsetneq \Lambda$. All terms are considered up to α-conversion (*i.e.* renaming of bound variables). The set of free variables of a term t is denoted by $\mathsf{fv}(t)$: t is *open* if $\mathsf{fv}(t) \neq \emptyset$, *closed* otherwise. Given $v \in \Lambda_v$, $t\{v/x\}$ denotes the term obtained by the *capture-avoiding substitution* of v for each free occurrence of x in the term t. Note that values are closed under substitution: if $v, v' \in \Lambda_v$ then $v\{v'/x\} \in \Lambda_v$.

One-hole contexts C are defined as usual, see Fig. 1. We use $C\langle t\rangle$ for the term obtained by the capture-allowing substitution of the term t for the hole $\langle\cdot\rangle$ in the context C. In Fig. 1 we define also a special kind of contexts, *balanced contexts B*.

Reductions in the shuffling calculus are defined in Fig. 1 as follows: given a *root-step* rule $\mapsto_{\mathsf{r}} \subseteq \Lambda \times \Lambda$, we define the r-*reduction* $\to_{\mathsf{r}}$ (resp. $\mathsf{r}^\flat$-*reduction* $\to_{\mathsf{r}^\flat}$) as the closure of $\mapsto_{\mathsf{r}}$ under contexts (resp. balanced contexts). The $\mathsf{r}^\flat$-reduction is non-deterministic and — because of balanced contexts — can reduce under abstractions, but it is "morally" *weak*: it reduces under a λ only when the λ is applied to an argument. Clearly, $\to_{\mathsf{sh}^\flat} \subsetneq \to_{\mathsf{sh}}$ since $\to_{\mathsf{sh}}$ can freely reduce under λ's.

The root-steps used in the shuffling calculus are $\mapsto_{\beta_v}$ (the reduction rule in Plotkin's λ_v), the commutation rules $\mapsto_{\sigma_1}$ and $\mapsto_{\sigma_3}$, and $\mapsto_{\sigma} := \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3}$ and $\mapsto_{\mathsf{sh}} := \mapsto_{\beta_v} \cup \mapsto_{\sigma}$. The side conditions for $\mapsto_{\sigma_1}$ and $\mapsto_{\sigma_3}$ in Fig. 1 can be always fulfilled by α-renaming. For any $\mathsf{r} \in \{\beta_v, \sigma_1, \sigma_3, \sigma, \mathsf{sh}\}$, if $t \mapsto_{\mathsf{r}} t'$ then t is a r-*redex* and t' is its r-*contractum*. A term of the shape $(\lambda x.t)u$ is a β-*redex*. Clearly, any β_v-redex is a β-redex but the converse does not hold: $(\lambda x.z)(yI)$ is a β-redex but not a β_v-redex. Redexes of different kind may *overlap*: for instance, the term $\Delta I \Delta$ is a σ_1-redex and contains the β_v-redex ΔI; the term $\Delta(I\Delta)(xI)$ is a σ_1-redex and contains the σ_3-redex $\Delta(I\Delta)$, which contains in turn the β_v-redex $I\Delta$.

From definitions in Fig. 1 it follows that $\to_{\mathsf{sh}} = \to_{\beta_v} \cup \to_{\sigma}$ and $\to_{\sigma} = \to_{\sigma_1} \cup \to_{\sigma_3}$, as well as $\to_{\mathsf{sh}^\flat} = \to_{\beta_v^\flat} \cup \to_{\sigma^\flat}$ and $\to_{\sigma^\flat} = \to_{\sigma_1^\flat} \cup \to_{\sigma_3^\flat}$. The *shuffling* (resp. *balanced shuffling*) *calculus* λ_{sh} (resp. $\lambda_{\mathsf{sh}}^\flat$) is the set Λ of terms endowed with the reduction $\to_{\mathsf{sh}}$ (resp. $\to_{\mathsf{sh}^\flat}$). The set Λ endowed with the reduction $\to_{\beta_v}$ is Plotkin's pure call-by-value λ-calculus λ_v [41], a sub-calculus of λ_{sh}.

Proposition 1 (Basic properties of reductions, [41, 14])**.** *The σ- and $\sigma^\flat$-reductions are confluent and strongly normalizing. The β_v-, $\beta_v^\flat$-,* sh- *and* $\mathsf{sh}^\flat$*-reductions are confluent.*

Example 2. Recall the terms t and u in Eq. (1): $t = (\lambda y.\Delta)(xI)\Delta \to_{\sigma_1^\flat} (\lambda y.\Delta\Delta)(xI) \to_{\beta_v^\flat} (\lambda y.\Delta\Delta)(xI) \to_{\beta_v^\flat} \dots$ and $u = \Delta((\lambda y.\Delta)(xI)) \to_{\sigma_3^\flat} (\lambda y.\Delta\Delta)(xI) \to_{\beta_v^\flat} (\lambda y.\Delta\Delta)(xI) \to_{\beta_v^\flat} \dots$ are the only possible sh-reduction paths from t and u respectively: t and u are not sh-normalizable and $t \simeq_{\mathsf{sh}} u$. But t and u are β_v-normal ($(\lambda y.\Delta)(xI)$ is a stuck β-redex) and different, so $t \not\simeq_{\beta_v} u$ by confluence of $\to_{\beta_v}$ (Prop. 1). Thus, $\simeq_{\beta_v} \subsetneq \simeq_{\mathsf{sh}}$.

Example 2 shows how σ-reduction shuffles constructors and moves stuck β-redex in order to unblock β_v-redexes which are hidden by the "hyper-sequential structure" of terms, avoiding "premature" normal forms. An alternative approach to circumvent the issue of stuck β-redexes is given by λ_{vsub}, the call-by-value λ-calculus with explicit substitutions introduced in [7], where hidden β_v-redexes are reduced using rules acting at a distance. In [4] it has been shown that λ_{vsub} and λ_{sh} can be embedded in each other preserving termination and divergence. Interestingly, both calculi are inspired by an analysis of Girard's "boring" call-by-value translation of λ-terms into linear logic proof-nets [25, 1] according to the linear recursive type $o = {!o} \multimap {!o}$, or equivalently $o = {!(o \multimap o)}$. In this translation, sh-reduction corresponds to cut-elimination, more precisely β_v-steps (resp. σ-steps) correspond to exponential (resp. multiplicative) cut-elimination steps; $\mathsf{sh}^\flat$-reduction corresponds to cut-elimination at depth 0.

Consider the two subsets of terms defined by mutual induction (notice that $\Lambda_a \subsetneq \Lambda_n \supsetneq \Lambda_v$):

$$a ::= xv \mid xa \mid an \quad \text{(set: } \Lambda_a) \qquad\qquad n ::= v \mid a \mid (\lambda x.n)a \quad \text{(set: } \Lambda_n).$$

Any $t \in \Lambda_a$ is neither a value nor a β-redex, but an open applicative term with a free "head variable".

Proposition 3 (Syntactic characterization on $\mathsf{sh}^\flat$-normal forms)**.** *Let t be a term:*

- *t is* $\mathsf{sh}^\flat$*-normal iff $t \in \Lambda_n$;*
- *t is* $\mathsf{sh}^\flat$*-normal and is neither a value nor a β-redex iff $t \in \Lambda_a$.*

Stuck β-redexes correspond to $\mathsf{sh}^\flat$-normal forms of the shape $(\lambda x.n)a$. As a consequence of Prop. 3, the behaviour of *closed* terms with respect to $\mathsf{sh}^\flat$-reduction (resp. $\beta_v^\flat$-reduction) is quite simple: either they diverge or they $\mathsf{sh}^\flat$-normalize (resp. $\beta_v^\flat$-normalize) to a closed value. Indeed:

Corollary 4 (Syntactic characterization of closed $\mathsf{sh}^\flat$- and $\beta_v^\flat$-normal forms)**.** *Let t be a closed term: t is* $\mathsf{sh}^\flat$*-normal iff t is $\beta_v^\flat$-normal iff t is a value iff $t = \lambda x.u$ for some term u with* $\mathsf{fv}(u) \subseteq \{x\}$.

3 A non-idempotent intersection type system

We recall the non-idempotent intersection type system introduced by Ehrhard [22] (nothing but the call-by-value version of de Carvalho's system R [15, 16]). We use it to characterize the (strong) normalizable terms for the reduction $\to_{\mathsf{sh}^\flat}$. *Types* are *positive* or *negative*, defined by mutual induction as follows:

$$\text{Negative Types:} \quad M,N ::= P \multimap Q \qquad\qquad \text{Positive Types:} \quad P,Q ::= [N_1,\dots,N_n] \ \ (\text{with } n \in \mathbb{N})$$

where $[N_1,\dots,N_n]$ is a (possibly empty) finite multiset of negative types; in particular the *empty multiset* $[\,]$ (obtained for $n = 0$) is the only atomic (positive) type. A positive type $[N_1,\dots,N_n]$ has to be intended as a conjunction $N_1 \wedge \cdots \wedge N_n$ of negative types $N_1,\dots,N_n$, for a commutative and associative conjunction connective $\wedge$ that is not idempotent and whose neutral element is $[\,]$.

The derivation rules for the non-idempotent intersection type system are in Fig. 2. In this typing system, *judgments* have the shape $\Gamma \vdash t : P$ where t is a term, P is a positive type and Γ is an *environment* (*i.e.* a

$$\frac{}{x\!:\!P \vdash x\!:\!P}\,\mathsf{ax} \qquad \frac{\Gamma \vdash t\!:\![P \multimap Q] \qquad \Gamma' \vdash u\!:\!P}{\Gamma \uplus \Gamma' \vdash tu\!:\!Q}\,@ \qquad \frac{\Gamma_1, x\!:\!P_1 \vdash t\!:\!Q_1 \quad \overset{n\in\mathbb{N}}{\cdots} \quad \Gamma_n, x\!:\!P_n \vdash t\!:\!Q_n}{\Gamma_1 \uplus \cdots \uplus \Gamma_n \vdash \lambda x.t\!:\![P_1 \multimap Q_1, \dots, P_n \multimap Q_n]}\,\lambda$$

Figure 2: Non-idempotent intersection type system for the shuffling calculus.

total function from variables to positive types whose domain $\mathsf{dom}(\Gamma) = \{x \mid \Gamma(x) \neq [\,]\}$ is finite). The *sum of environments* $\Gamma \uplus \Delta$ is defined pointwise via multiset sum: $(\Gamma \uplus \Delta)(x) = \Gamma(x) \uplus \Delta(x)$. An environment Γ such that $\mathsf{dom}(\Gamma) \subseteq \{x_1, \dots, x_n\}$ with $x_i \neq x_j$ and $\Gamma(x_i) = P_i$ for all $1 \leq i \neq j \leq k$ is often written as $\Gamma = x_1 : P_1, \dots, x_n : P_k$. In particular, Γ and $\Gamma, x : [\,]$ (where $x \notin \mathsf{dom}(\Gamma)$) are the same environment; and $\vdash t : P$ stands for the judgment $\Gamma \vdash t : P$ where Γ is the *empty environment*, *i.e.* $\mathsf{dom}(\Gamma) = \emptyset$ (that is, $\Gamma(x) = [\,]$ for any variable x). Note that the sum of environments $\uplus$ is commutative, associative and its neutral element is the empty environment: given an environment Γ, one has $\Gamma \uplus \Delta = \Gamma$ iff $\mathsf{dom}(\Delta) = \emptyset$. The notation $\pi \rhd \Gamma \vdash t : P$ means that π is a derivation with conclusion the judgment $\Gamma \vdash t : P$. We write $\pi \rhd t$ if π is such that $\pi \rhd \Gamma \vdash t : P$ for some environment Γ and positive type P.

It is worth noticing that the type system in Fig. 2 is *syntax oriented*: for each type judgment J there is a *unique* derivation rule whose conclusion matches the judgment J.

The *size* $|\pi|$ of a type derivation π is just the the number of @ rules in π. Note that judgments play no role in the size of a derivation.

Example 5. Let $I = \lambda x.x$. The derivations (typing II and I with same type and same environment)

$$\pi_{II} = \frac{\dfrac{\dfrac{}{x\colon [\,] \vdash x\colon [\,]}\,\mathsf{ax}}{\vdash I\colon [[\,] \multimap [\,]]}\,\lambda \qquad \dfrac{}{\vdash I\colon [\,]}\,\lambda}{\vdash II\colon [\,]}\,@ \qquad\qquad \pi_I = \frac{}{\vdash I\colon [\,]}\,\lambda$$

are such that $|\pi_{II}| = 1$ and $|\pi_I| = 0$. Note that $II \to_{\mathsf{sh}^\flat} I$ and $|\pi_{II}| = |\pi_I| + 1$.

The following lemma (whose proof is quite technical) will play a crucial role to prove the substitution lemma (Lemma 7) and the subject reduction (Prop. 8) and expansion (Prop. 10).

Lemma 6 (Judgment decomposition for values)**.** *Let* $v \in \Lambda_v$*,* Δ *be an environment, and* $P_1, \dots, P_p$ *be positive types (for some* $p \in \mathbb{N}$*). There is a derivation* $\pi \rhd \Delta \vdash v\colon P_1 \uplus \cdots \uplus P_p$ *iff for all* $1 \leq i \leq p$ *there are an environment* Δ_i *and a derivation* $\pi_i \rhd \Delta_i \vdash v\colon P_i$ *such that* $\Delta = \biguplus_{i=1}^{p} \Delta_i$*. Moreover,* $|\pi| = \sum_{i=1}^{p} |\pi_i|$*.*

The left-to-right direction of Lemma 6 means that, given $\pi \rhd \Delta \vdash v\colon P$, for *every* $p \in \mathbb{N}$ and *every* decomposition of the positive type P into a multiset sum of positive types $P_1, \dots, P_p$, there are environments $\Delta_1, \dots, \Delta_p$ such that $\Delta_i \vdash v\colon P_i$ is derivable for all $1 \leq i \leq p$.

Lemma 7 (Substitution)**.** *Let* $t \in \Lambda$ *and* $v \in \Lambda_v$*. If* $\pi \rhd \Gamma, x\colon P \vdash t\colon Q$ *and* $\pi' \rhd \Delta \vdash v\colon P$*, then there exists* $\pi'' \rhd \Gamma \uplus \Delta \vdash t\{v/x\}\colon Q$ *such that* $|\pi''| = |\pi| + |\pi'|$*.*

We can now prove the subject reduction, with a quantitative flavour about the size of type derivations in order to extract information about the execution time.

Proposition 8 (Quantitative balanced subject reduction)**.** *Let* $t, t' \in \Lambda$ *and* $\pi \rhd \Gamma \vdash t\colon Q$*.*

1. Shrinkage under $\beta_v^\flat$-step: *If* $t \to_{\beta_v^\flat} t'$ *then* $|\pi| > 0$ *and there exists a derivation* π' *with conclusion* $\Gamma \vdash t'\colon Q$ *such that* $|\pi'| = |\pi| - 1$*.*

2. Size invariance under $\sigma^\flat$-step: *If* $t \to_{\sigma^\flat} t'$ *then* $|\pi| > 0$ *and there exists a derivation* π' *with conclusion* $\Gamma \vdash t'\colon Q$ *such that* $|\pi'| = |\pi|$*.*

In Prop. 8, the fact that $\to_{\mathsf{sh}^\flat}$ does not reduce under λ's is crucial to get the quantitative information, otherwise one can have a term t such that every derivation $\pi \rhd \Gamma \vdash t\colon P$ is such that $|\pi| = 0$ (and then there is no derivation π' with conclusion $\Gamma \vdash t'\colon P$ such that $|\pi| = |\pi'| - 1$): this is the case, for example, for $t = \lambda x.\delta\delta \to_{\beta_v} t$. This shows that the quantitative study for evaluation reducing under λ's is subtler.

In order to prove the quantitative subject expansion (Prop. 10), we first need the following technical lemma stating the commutation of abstraction with abstraction and application.

Lemma 9 (Abstraction commutation)**.**

1. Abstraction vs. abstraction: *Let* $k \in \mathbb{N}$. *If* $\pi \rhd \Delta \vdash \lambda y.(\lambda x.t)v\colon \biguplus_{i=1}^{k}[P_i' \multimap P_i]$ *and* $y \notin \mathsf{fv}(v)$, *then there is* $\pi' \rhd \Delta \vdash (\lambda x.\lambda y.t)v\colon \biguplus_{i=1}^{k}[P_i' \multimap P_i]$ *such that* $|\pi'| = |\pi| + 1 - k$.

2. Application vs. abstraction: *If* $\pi \rhd \Delta \vdash ((\lambda x.t)v)((\lambda x.u)v)\colon P$ *then there exists a derivation* $\pi' \rhd \Delta \vdash (\lambda x.tu)v\colon P$ *such that* $|\pi'| = |\pi| - 1$.

Proposition 10 (Quantitative balanced subject expansion)**.** *Let* $t, t' \in \Lambda$ *and* $\pi' \rhd \Gamma \vdash t'\colon Q$.

1. Enlargement under anti-$\beta_v^\flat$-step: *If* $t \to_{\beta_v^\flat} t'$ *then there is* $\pi \rhd \Gamma \vdash t\colon Q$ *with* $|\pi| = |\pi'| + 1$.

2. Size invariance under anti-$\sigma^\flat$-step: *If* $t \to_{\sigma^\flat} t'$ *then* $|\pi'| > 0$ *and there is* $\pi \rhd \Gamma \vdash t\colon Q$ *with* $|\pi| = |\pi'|$.

Actually, subject reduction and expansion hold for the whole sh-reduction $\to_{\mathsf{sh}}$, not only for the balanced sh-reduction $\to_{\mathsf{sh}^\flat}$. The drawback for $\to_{\mathsf{sh}}$ is that the quantitative information about the size of the derivation is lost in the case of a β_v-step, see the comments just after Prop. 8 and Lemma 12.

Lemma 11 (Subject reduction)**.** *Let* $t, t' \in \Lambda$ *and* $\pi \rhd \Gamma \vdash t\colon Q$.

1. Shrinkage under β_v-step: *If* $t \to_{\beta_v} t'$ *then there is* $\pi' \rhd \Gamma \vdash t'\colon Q$ *with* $|\pi| \geq |\pi'|$.

2. Size invariance under σ-step: *If* $t \to_\sigma t'$ *then there is* $\pi' \rhd \Gamma \vdash t'\colon Q$ *such that* $|\pi| = |\pi'|$.

Lemma 12 (Subject expansion)**.** *Let* $t, t' \in \Lambda$ *and* $\pi' \rhd \Gamma \vdash t'\colon Q$.

1. Enlargement under anti-β_v-step: *If* $t \to_{\beta_v} t'$ *then there is* $\pi \rhd \Gamma \vdash t\colon Q$ *with* $|\pi| \geq |\pi'|$.

2. Size invariance under anti-σ-step: *If* $t \to_\sigma t'$ *then there is* $\pi \rhd \Gamma \vdash t\colon Q$ *such that* $|\pi| = |\pi'|$.

In Lemmas 11.1 and 12.1 it is impossible to estimate more precisely the relationship between $|\pi|$ and $|\pi'|$. Indeed, Ex. 5 has shown that there are $\pi_I \rhd y\colon [\,] \vdash I\colon [\,]$ and $\pi_{II} \rhd y\colon [\,] \vdash II\colon [\,]$ such that $|\pi_I| = 0$ and $|\pi_{II}| = 1$ (where $I = \lambda x.x$). So, given $k \in \mathbb{N}$, consider the derivations $\pi_k \rhd \vdash \lambda y.II\colon [[\,] \multimap [\,], \overset{k}{\ldots}, [\,] \multimap [\,]]$ and $\pi_k' \rhd \vdash \lambda y.I\colon [[\,] \multimap [\,], \overset{k}{\ldots}, [\,] \multimap [\,]]$ below:

$$\pi_n = \dfrac{\dfrac{\vdots\,\pi_{II}}{y\colon [\,] \vdash II\colon [\,]} \quad \overset{k}{\ldots} \quad \dfrac{\vdots\,\pi_{II}}{y\colon [\,] \vdash II\colon [\,]}}{\vdash \lambda y.II\colon [[\,] \multimap [\,], \overset{k}{\ldots}, [\,] \multimap [\,]]}\,\lambda \qquad \pi_n' = \dfrac{\dfrac{\vdots\,\pi_{I}}{y\colon [\,] \vdash I\colon [\,]} \quad \overset{k}{\ldots} \quad \dfrac{\vdots\,\pi_{I}}{y\colon [\,] \vdash I\colon [\,]}}{\vdash \lambda y.I\colon [[\,] \multimap [\,], \overset{k}{\ldots}, [\,] \multimap [\,]]}\,\lambda$$

Clearly, $\lambda y.II \to_{\mathsf{sh}} \lambda y.I$ (but $\lambda y.II \not\to_{\mathsf{sh}^\flat} \lambda y.I$) and the π_k' (resp. π_k) is the only derivation typing $\lambda y.I$ (resp. $\lambda y.II$) with the same type and environment as π_k (resp. π_k'). One has $|\pi_k| = k \cdot |\pi_{II}| = k$ and $|\pi_k'| = k \cdot |\pi_I| = 0$, thus the difference of size of the derivations π_k and π_k' can be arbitrarely large (since $k \in \mathbb{N}$); in particular $|\pi_0| = |\pi_0'|$, so for $k = 0$ the size of derivations does not even strictly decrease.

4 Relational semantics: qualitative results

Lemmas 11 and 12 have an important consequence: the non-idempotent intersection type system of Fig. 2 defines a denotational model for the shuffling calculus λ_{sh} (Thm. 14 below).

Definition 13 (Suitable list of variables for a term, semantics of a term)**.** Let $t \in \Lambda$ and let $x_1, \dots, x_k$ be pairwise distinct variables, for some $k \in \mathbb{N}$.

If $\mathsf{fv}(t) \subseteq \{x_1, \dots, x_k\}$, then we say that the list $\vec{x} = (x_1, \dots, x_k)$ is *suitable for t*.

If $\vec{x} = (x_1, \dots, x_k)$ is suitable for t, the (*relational*) *semantics*, or *interpretation*, *of t for $\vec{x}$* is

$$[\![t]\!]_{\vec{x}} = \{((P_1, \dots, P_k), Q) \mid \exists \pi \rhd x_1 : P_1, \dots, x_k : P_k \vdash t : Q\}.$$

Essentially, the semantics of a term t for a suitable list $\vec{x}$ of variables is the set of judgments for $\vec{x}$ and t that can be derived in the non-idempotent intersection type system of Fig. 2.

If we identify the negative type $P \multimap Q$ with the pair (P, Q) and if we set $\mathscr{U} := \bigcup_{k \in \mathbb{N}} \mathscr{U}_k$ where:

$$\mathscr{U}_0 := \emptyset \qquad \mathscr{U}_{k+1} := \mathscr{M}_{\mathrm{f}}(\mathscr{U}_k) \times \mathscr{M}_{\mathrm{f}}(\mathscr{U}_k) \qquad (\mathscr{M}_{\mathrm{f}}(X) \text{ is the set of finite multisets over the set } X)$$

then, for any $t \in \Lambda$ and any suitable list $\vec{x} = (x_1, \dots, x_k)$ for t, one has $[\![t]\!]_{\vec{x}} \subseteq \mathscr{M}_{\mathrm{f}}(\mathscr{U})^k \times \mathscr{M}_{\mathrm{f}}(\mathscr{U})$; in particular, if t is closed and $\vec{x} = (\,)$, then $[\![t]\!] = \{Q \mid \exists \pi \rhd\ \vdash t : Q\} \subseteq \mathscr{M}_{\mathrm{f}}(\mathscr{U})$ (up to an obvious isomorphism). Note that $\mathscr{U} = \mathscr{M}_{\mathrm{f}}(\mathscr{U}) \times \mathscr{M}_{\mathrm{f}}(\mathscr{U})$: [22, 14] proved that the latter identity is enough to have a denotational model for λ_{sh}. We can also prove it explicitly using Lemmas 11 and 12.

Theorem 14 (Invariance under sh-equivalence)**.** *Let $t, u \in \Lambda$, let $k \in \mathbb{N}$ and let $\vec{x} = (x_1, \dots, x_k)$ be a suitable list of variables for t and u. If $t \simeq_{\mathsf{sh}} u$ then $[\![t]\!]_{\vec{x}} = [\![u]\!]_{\vec{x}}$.*

An interesting property of relational semantics is that all $\mathsf{sh}^\flat$-normal forms have a non-empty interpretation (Lemma 15). To prove that we use the syntactic characterization of $\mathsf{sh}^\flat$-normal forms (Prop. 3). Note that a stronger statement (Lemma 15.1) is required for $\mathsf{sh}^\flat$-normal forms belonging to Λ_a, in order to handle the case where the $\mathsf{sh}^\flat$-normal form is a β-redex.

Lemma 15 (Semantics and typability of $\mathsf{sh}^\flat$-normal forms)**.** *Let t be a term, let $k \in \mathbb{N}$ and let $\vec{x} = (x_1, \dots, x_k)$ be a list of variables suitable for t.*

1. *If $t \in \Lambda_a$ then for every positive type Q there exist positive types $P_1, \dots, P_k$ and a derivation $\pi \rhd x_1 : P_1, \dots, x_k : P_k \vdash t : Q$.*
2. *If $t \in \Lambda_n$ then there are positive types $Q, P_1, \dots, P_k$ and a derivation $\pi \rhd x_1 : P_1, \dots, x_k : P_k \vdash t : Q$.*
3. *If t is $\mathsf{sh}^\flat$-normal then $[\![t]\!]_{\vec{x}} \neq \emptyset$.*

A consequence of Prop. 8 (and Thm. 14 and Lemma 15) is a qualitative result: a semantic and logical (if we consider our non-idempotent type system as a logical framework) characterization of (strong) $\mathsf{sh}^\flat$-normalizable terms (Thm. 16). In this theorem, the main equivalences are between Points 1, 3 and 5, already proven in [14] using different techniques. Points 2 and 4 can be seen as “intermediate stages” in the proof of the main equivalences, which are informative enough to deserve to be explicitely stated.

Theorem 16 (Semantic and logical characterization of $\mathsf{sh}^\flat$-normalization)**.** *Let $t \in \Lambda$ and let $\vec{x} = (x_1, \dots, x_k)$ be a suitable list of variables for t. The following are equivalent:*

1. Normalizability: *t is $\mathsf{sh}^\flat$-normalizable;*
2. Completeness: *$t \simeq_{\mathsf{sh}} u$ for some $\mathsf{sh}^\flat$-normal $u \in \Lambda$;*
3. Adequacy: *$[\![t]\!]_{\vec{x}} \neq \emptyset$;*

4. Derivability: *there is a derivation $\pi \rhd x_1 : P_1, \dots, x_k : P_k \vdash t : Q$ for some positive types $P_1, \dots, P_k, Q$;*
5. Strong normalizabilty: *t is strongly $\mathsf{sh}^\flat$-normalizable.*

As implication (5)⇒(1) is trivial, the proof of Thm. 16 follows the structure (1)⇒(2)⇒(3)⇒(4)⇒(5): essentially, non-idempotent intersection types are used to prove that normalization implies strong normalization for $\mathsf{sh}^\flat$-reduction. Equivalence (5)⇔(1) means that normalization and strong normalization are equivalent for $\mathsf{sh}^\flat$-reduction, thus in studying the termination of $\mathsf{sh}^\flat$-reduction no intricacy arises from its non-determinism. Although $\to_{\mathsf{sh}^\flat}$ does not evaluate under λ's, this result is not trivial because $\to_{\mathsf{sh}^\flat}$ does not enjoy any form of (quasi-)diamond property, as we show in Ex. 23 below. Equivalence (1)⇔(2) says that $\mathsf{sh}^\flat$-reduction is complete with respect to sh-equivalence to get $\mathsf{sh}^\flat$-normal forms; in particular, this entails that every sh-normalizable term is $\mathsf{sh}^\flat$-normalizable. Equivalence (1)⇔(2) is the analogue of a well-known theorem [8, Thm. 8.3.11] for ordinary (*i.e.* call-by-name) λ-calculus relating head β-reduction and β-equivalence: this corroborates the idea that $\mathsf{sh}^\flat$-reduction is the "head reduction" in a call-by-value setting, despite its non-determinism. The equivalence (3)⇔(4) holds by definition of relational semantics.

Implication (1)⇒(3) (or equivalently (1)⇒(4), *i.e.* "normalizable ⇒ typable") does not hold in Plotkin's λ_v: indeed, the (open) terms t and u in Eq. (1) (see also Ex. 2) are β_v-normal (because of a stuck β-redex) but $[\![t]\!]_x = \emptyset = [\![u]\!]_x$. Equivalences such as the ones in Thm. 16 hold in a call-by-value setting provided that β_v-reduction is extended, *e.g.* by adding σ-reduction. In [4], λ_{sh} is proved to be termination equivalent to other extensions of λ_v (in the framework Open Call-by-Value, where evaluation is call-by-value and weak, on possibly open terms) such as the fireball calculus [45, 27, 2] and the value substitution calculus [7], so Thm. 16 is a general result *characterizing termination in those calculi* as well.

Lemma 17 (Uniqueness of the derivation with empty types; Semantic and logical characterization of values)**.** *Let $t \in \Lambda$ be $\mathsf{sh}^\flat$-normal.*

1. *If $\pi \rhd \vdash t : [\,]$ and $\pi' \rhd \Gamma \vdash t : [\,]$, then $t \in \Lambda_v$, $|\pi| = 0$, $\mathsf{dom}(\Gamma) = \emptyset$ and $\pi = \pi'$. More precisely, π consists of a rule ax if t is a variable, otherwise t is an abstraction and π consists of a 0-ary rule λ.*
2. *Given a list $\vec{x} = (x_1, \dots, x_k)$ of variables suitable for t, the following are equivalent:*
   - *(a) t is a value;*
   - *(b) $(([\,], \overset{k}{\dots}, [\,]), [\,]) \in [\![t]\!]_{\vec{x}}$;*
   - *(c) there exists $\pi \rhd \vdash t : [\,]$;*
   - *(d) there exists $\pi \rhd t$ such that $|\pi| = 0$.*

Qualitatively, Lemma 17 allows us to refine the semantic and logical characterization given by Thm. 16 for a specific class of terms: the *valuable* ones, *i.e.* the terms that $\mathsf{sh}^\flat$-normalize to a value. Valuable terms are all and only the terms whose semantics contains a specific element: the point with only empty types.

Proposition 18 (Logical and semantic characterization of valuability)**.** *Let t be a term and $\vec{x} = (x_1, \dots, x_k)$ be a suitable list of variables for t. The following are equivalent:*

1. Valuability: *t is $\mathsf{sh}^\flat$-normalizable and the $\mathsf{sh}^\flat$-normal form of t is a value;*
2. Empty point in the semantics: $(([\,], \overset{k}{\dots}, [\,]), [\,]) \in [\![t]\!]_{\vec{x}}$*;*
3. Derivability with empty types: *there exists a derivation $\pi \rhd \vdash t : [\,]$.*

5 The quantitative side of type derivations

By the quantitative subject reduction (Prop. 8), the size of *any* derivation typing a ($\mathsf{sh}^\flat$-normalizable) term t is an upper bound on the number $\mathsf{leng}_{\beta_v^\flat}(d)$ of $\beta_v^\flat$-steps in *any* $\mathsf{sh}^\flat$-normalizing reduction sequence d from t, since the size of a type derivation decreases by 1 after each $\beta_v^\flat$-step, and does not change after each $\sigma^\flat$-step.

Corollary 19 (Upper bound on the number of $\beta_v^\flat$-steps)**.** *Let t be a* $\mathsf{sh}^\flat$*-normalizable term and t_0 be its* $\mathsf{sh}^\flat$*-normal form. For any reduction sequence $d\colon t \to_{\mathsf{sh}^\flat}^* t_0$ and any $\pi \rhd t$,* $\mathrm{leng}_{\beta_v^\flat}(d) \leq |\pi|$.

In order to extract from a type derivation the *exact* number of $\beta_v^\flat$-steps to reach the $\mathsf{sh}^\flat$-normal form, we have to take into account also the size of derivations of $\mathsf{sh}^\flat$-normal forms. Indeed, by Lemma 17.2, $\mathsf{sh}^\flat$-normal forms that are not values admit only derivations with sizes greater than 0. The sizes of type derivations of a $\mathsf{sh}^\flat$-normal form t are related to a special kind of size of t that we now define.

The *balanced size* of a term t, denoted by $|t|_\flat$, is defined by induction on t as follows ($v \in \Lambda_v$):

$$|v|_\flat = 0 \qquad\qquad |tu|_\flat = \begin{cases} |s|_\flat + |u|_\flat + 1 & \text{if } t = \lambda x.s \\ |t|_\flat + |u|_\flat + 1 & \text{otherwise.} \end{cases}$$

So, the balanced size of a term t is the number of applications occurring in t under a balanced context, *i.e.* the number of pairs (u,s) such that $t = B\langle us\rangle$ for some balanced context B. For instance, $|(\lambda x.yy)(zz)|_\flat = 3$ and $|(\lambda x.\lambda x'.yy)(zz)|_\flat = 2$. The following lemma can be seen as a quantitative version of Lemma 15.

Lemma 20 (Relationship between sizes of normal forms and derivations)**.** *Let $t \in \Lambda$.*

1. *If t is* $\mathsf{sh}^\flat$*-normal then* $|t|_\flat = \min\{|\pi| \mid \pi \rhd t\}$.
2. *If t is a value then* $|t|_\flat = \min\{|\pi| \mid \pi \rhd t\} = 0$.

Thus, the balanced size of a $\mathsf{sh}^\flat$-normal form n equals the minimal size of the type derivation of n.

Proposition 21 (Exact number of $\beta_v^\flat$-steps)**.** *Let t be a* $\mathsf{sh}^\flat$*-normalizable term and t_0 be its* $\mathsf{sh}^\flat$*-normal form. For every reduction sequence $d\colon t \to_{\mathsf{sh}^\flat}^* t_0$ and every $\pi \rhd t$ and $\pi_0 \rhd t_0$ such that* $|\pi| = \min\{|\pi'| \mid \pi' \rhd t\}$ *and* $|\pi_0| = \min\{|\pi_0'| \mid \pi_0' \rhd t_0\}$*, one has*

$$\mathrm{leng}_{\beta_v^\flat}(d) = |\pi| - |t_0|_\flat = |\pi| - |\pi_0|\,. \tag{2}$$

If moreover t_0 is a value, then $\mathrm{leng}_{\beta_v^\flat}(d) = |\pi|$.

In particular, Eq. (2) implies that for *any* reduction sequence $d\colon t \to_{\mathsf{sh}^\flat}^* t_0$ and *any* $\pi \rhd t$ and $\pi_0 \rhd t_0$ such that $|\pi_0| = \min\{|\pi_0'| \mid \pi_0' \rhd t_0\}$, one has $\mathrm{leng}_{\beta_v^\flat}(d) \leq |\pi| - |t_0|_\flat = |\pi| - |\pi_0|$, since $|\pi| \geq \min\{|\pi'| \mid \pi' \rhd t\}$.

Prop. 21 could seem slightly disappointinig: it allows us to know the exact number of $\beta_v^\flat$-steps of a $\mathsf{sh}^\flat$-normalizing reduction sequence from t only if we already know the $\mathsf{sh}^\flat$-normal form t_0 of t (or the minimal derivation of t_0), which essentially means that we have to perform the reduction sequence in order to know the exact number of its $\beta_v^\flat$-steps. However, Prop. 21 says also that this limitation is circumvented in the case t $\mathsf{sh}^\flat$-reduces to a value. Moreover, a notable and immediate consequence of Prop. 21 is:

Corollary 22 (Same number of $\beta_v^\flat$-steps)**.** *Let t be a* $\mathsf{sh}^\flat$*-normalizable term and t_0 be its* $\mathsf{sh}^\flat$*-normal form. For all reduction sequences $d\colon t \to_{\mathsf{sh}^\flat}^* t_0$ and $d'\colon t \to_{\mathsf{sh}^\flat}^* t_0$, one has* $\mathrm{leng}_{\beta_v^\flat}(d) = \mathrm{leng}_{\beta_v^\flat}(d')$.

Even if $\mathsf{sh}^\flat$-reduction is weak, in the sense that it does not reduce under λ's, Cor. 22 is not obvious at all, since the rewriting theory of $\mathsf{sh}^\flat$-reduction is not quite elegant, in particular it does not enjoy any form of (quasi-)diamond property because of σ-reduction, as shown by the following example.

Example 23. Let $t := (\lambda y.y')(\Delta(xI))I$: one has $u := (\lambda y.y')(\Delta(xI))\ {}_{\sigma_1^\flat}\!\!\leftarrow t \to_{\sigma_3^\flat} (\lambda z.(\lambda y.y')(zz))(xI)I =: s$ and the only way to join this critical pair is by performing *one* $\sigma_3^\flat$-step from u and *two* $\sigma_1^\flat$-steps from s, so that $u \to_{\sigma_3^\flat} (\lambda z.(\lambda y.y'I)(zz))(xI)\ {}_{\sigma_1^\flat}\!\!\leftarrow (\lambda z.(\lambda y.y')(zz)I)(xI)\ {}_{\sigma_1^\flat}\!\!\leftarrow s$. Since each $\sigma^\flat$-step can create a new β_v-redex in a balanced context (as shown in Ex. 2), *a priori* there is no evidence that Cor. 22 should hold.

Cor. 22 allows us to define the following function $\mathrm{leng}_{\beta_v}\colon \Lambda \to \mathbb{N} \cup \{\infty\}$

$$\mathrm{leng}_{\beta_v^\flat}(t) = \begin{cases} \mathrm{leng}_{\beta_v^\flat}(d) & \text{if there is a } \mathsf{sh}^\flat\text{-normalizing reduction sequence } d \text{ from } t; \\ \infty & \text{otherwise.} \end{cases}$$

In other words, in λ_{sh} we can univocally associate with every term the number of $\beta_v^\flat$-steps needed to reach its $\mathsf{sh}^\flat$-normal form, if any (the infinity ∞ is associated with non-$\mathsf{sh}^\flat$-normalizable terms). The characterization of $\mathsf{sh}^\flat$-normalization given in Thm. 16 allows us to determine through semantic or logical means if the value of $\mathrm{leng}_{\beta_v^\flat}(t)$ is a finite number or not.

Quantitatively, via Lemma 17 we can simplify the way to compute the number of $\beta_v^\flat$-steps to reach the $\mathsf{sh}^\flat$-normal form of a valuable (*i.e.* that reduces to a value) term t, using only a specific type derivation of t.

Theorem 24 (Exact number of $\beta_v^\flat$-steps for valuables)**.** *If* $t \to_{\mathsf{sh}^\flat}^* v \in \Lambda_v$ *then* $\mathrm{leng}_{\beta_v^\flat}(t) = |\pi|$ *for* $\pi \rhd \vdash t\colon [\,]$.

Prop. 18 and Thm. 24 provide a procedure to determine if a term t $\mathsf{sh}^\flat$-normalizes to a value and, in case, how many $\beta_v^\flat$-steps are needed to reach its $\mathsf{sh}^\flat$-normal form (this number does not depend on the reduction strategy according to Cor. 22), considering only the term t and without performing any $\mathsf{sh}^\flat$-step:

1. check if there is a derivation π with empty types, *i.e.* $\pi \rhd \vdash t\colon [\,]$;
2. if it is so (*i.e.* if t $\mathsf{sh}^\flat$-normalize to a value, according to Prop. 18), compute the size $|\pi|$.

Remind that, according to Cor. 4, any closed term either is not $\mathsf{sh}^\flat$-normalizable, or it $\mathsf{sh}^\flat$-normalizes to a (closed) value. So, this procedure completely determines (qualitatively and quantitatively) the behavior of closed terms with respect to $\mathsf{sh}^\flat$-reduction (and to $\beta_v^\flat$-reduction, as we will see in Sect. 6).

6 Conclusions

Back to Plotkin's λ_v. The shuffling calculus λ_{sh} can be used to prove some properties of Plotkin's call-by-value λ-calculus λ_v (whose only reduction rule is $\to_{\beta_v}$) *restricted to closed terms*. This is an example of how the study of some properties of a framework (in this case, λ_v) can be naturally done in a more general framework (in this case, λ_{sh}). It is worth noting that λ_v with only closed terms is an interesting fragment: it represents the core of many functional programming languages, such as OCaml.

The starting point is Cor. 4, which says that, in the closed setting with weak reduction, normal forms for λ_{sh} and λ_v coincide: they are all and only closed values. We can then reformulate Thm. 16 and Prop. 18 as a semantic and logical characterization of $\beta_v^\flat$-normalization in Plotkin's λ_v *restricted to closed terms*.

Theorem 25 (Semantic and logical characterization of $\beta_v^\flat$-normalization in the closed case)**.** *Let t be a closed term. The following are equivalent:*

1. Normalizability: *t is $\beta_v^\flat$-normalizable;*
2. Valuability: *$t \to_{\beta_v^\flat}^* v$ for some closed value v;*
3. Completeness: *$t \simeq_{\beta_v} v$ for some closed value v;*
4. Adequacy: *$[\![t]\!]_{\vec{x}} \neq \emptyset$ for any list $\vec{x} = (x_1,\dots,x_k)$ (with $k \in \mathbb{N}$) of pairwise distinct variables;*
5. Empty point: *$(([\,],\overset{k}{\ldots},[\,]),[\,]) \in [\![t]\!]_{\vec{x}}$ for any list $\vec{x} = (x_1,\dots,x_k)$ ($k \in \mathbb{N}$) of pairwise distinct variables;*
6. Derivability with empty types: *there exists a derivation $\pi \rhd \vdash t\colon [\,]$;*
7. Derivability: *there exists a derivation $\pi \rhd \vdash t\colon Q$ for some positive type Q;*

8. Strong normalizabilty: *t is strongly $\beta_v^\flat$-normalizable.*

We have already seen on pp. 64–65 that Thm. 25 does not hold in λ_v with open terms: closure is crucial.

Thm. 25 entails that a closed term is $\mathsf{sh}^\flat$-normalizable iff it is $\beta_v^\flat$-normalizable iff it $\beta_v^\flat$-reduces to a closed value. Thus, Cor. 22 and Thm. 24 can be reformulated for λ_v *restricted to closed terms* as follows.

Corollary 26 (Same number of $\beta_v^\flat$-steps)**.** *Let t be a closed $\beta_v^\flat$-normalizable term and t_0 be its $\beta_v^\flat$-normal form. For all reduction sequences $d\colon t \to_{\beta_v^\flat}^* t_0$ and $d'\colon t \to_{\beta_v^\flat}^* t_0$, one has* $\mathrm{leng}_{\beta_v^\flat}(d) = \mathrm{leng}_{\beta_v^\flat}(d')$.

Theorem 27 (Number of $\beta_v^\flat$-steps)**.** *If t is closed and $\beta_v^\flat$-normalizable, then* $\mathrm{leng}_{\beta_v^\flat}(t) = |\pi|$ *for $\pi \rhd\, \vdash t\colon [\,]$.*

Clearly, the procedure sketched on p. 67, when applied to a *closed* term t, determines if t $\beta_v^\flat$-normalizes and, in case, how many $\beta_v^\flat$-steps are needed to reach its $\beta_v^\flat$-normal form.

Towards a semantic measure. In order to get a truly semantic measure of the execution time in the shuffling calculus λ_{sh}, we should first be able to give an *upper bound* to the number of $\beta_v^\flat$-steps in a $\mathsf{sh}^\flat$-reduction looking only at the semantics of terms. Therefore, we need to define a notion of size for the elements of the semantics of terms. The most natural approach is the following. For any positive type $P = [P_1 \multimap Q_1, \dots, P_k \multimap Q_k] \in \mathscr{M}_{\mathrm{f}}(\mathscr{U})$ (with $k \in \mathbb{N}$), the *size of* P is $|P| = k + \sum_{i=1}^{k}(|P_i| + |Q_i|)$. So, the size of a positive type P is the number of occurrences of $\multimap$ in P; in particular, $|[\,]| = 0$. For any $((P_1,\dots,P_n),Q) \in \mathscr{M}_{\mathrm{f}}(\mathscr{U})^k \times \mathscr{M}_{\mathrm{f}}(\mathscr{U})$ (with $k \in \mathbb{N}$), the *size of* $((P_1,\dots,P_k),Q)$ is $|((P_1,\dots,P_k),Q)| = |Q| + \sum_{i=1}^{k}|P_i|$.

The approach of [16, 17] relies on a crucial lemma to find an upper bound (and hence the exact length) of the execution time: it relates the size of a type derivation to the size of its conclusion, for a *normal* term/proof-net. In λ_{sh} this lemma should claim that "For every sh-*normal* form t, if $\pi \rhd x_1\colon P_1,\dots,x_k\colon P_k \vdash t\colon Q$ then $|\pi| \leq |((P_1,\dots,P_k),Q)|$". Unfortunately, in λ_{sh} this property is false!

Example 28. Let $t \coloneqq (\lambda x.x)(yy)$, which is a sh-normal form. Consider the derivation

$$\pi \coloneqq \dfrac{\dfrac{\dfrac{}{x\colon [\,] \vdash x\colon [\,]}\,\mathsf{ax}}{\vdash \lambda x.x\colon [[\,] \multimap [\,]]}\,\lambda \qquad \dfrac{\dfrac{}{x\colon [[\,] \multimap [\,]] \vdash x\colon [[\,] \multimap [\,]]}\,\mathsf{ax} \qquad \dfrac{}{x\colon [\,] \vdash x\colon [\,]}\,\mathsf{ax}}{y\colon [[\,] \multimap [\,]] \vdash yy\colon [\,]}\,@}{y\colon [[\,] \multimap [\,]] \vdash (\lambda x.x)(yy)\colon [\,]}\,@ \;.$$

Then, $|\pi| = 2 > 1 = |(([[\,] \multimap [\,]], [\,])|$, which provides a counterexample to the property demanded above.

We conjecture that in order to overcome this counterexample (and to successfully follow the method of [16, 17] to get a purely semantic measure of the execution time) we should change the syntax and the operational semantics of our calculus, always remaining in a call-by-value setting equivalent (from the termination point of view) to λ_{sh} and the other calculi studied in [4]. Intuitively, in Ex. 28 t contains one application — $(\lambda x.x)(yy)$ — that is a stuck β-redex and is the source of one "useless" instance of the rule @ in π. The idea for the new calculus is to "fire" a stuck β-redex $(\lambda x.t)u$ without performing the substitution $t\{u/x\}$ (as u might not be a value), but just creating an explicit substitution $t[u/x]$ that removes the application but "stores" the stuck β-redex. Such a calculus has been recently introduced in [5].

Related work. This work has been presented at the workshop ITRS 2018. Later, the author further investigated this topic with Beniamino Accattoli in [5], where we applied the same type system (and hence the same relational semantics) to a different call-by-value calculus with weak evaluation, λ_{fire}. The techniques used in both papers are similar (but not identical), some differences are due to the distinct calculi the type system is applied to. Some results are analogous: semantic and logical characterization of termination, extraction of quantitative information from type derivations. In [5] we focused on an abstract characterization of the type derivations that provide an exact bound on the number of steps to reach the normal form. Here, the semantic and logical characterization of termination is more informative than in [5] because the reduction in λ_{sh} is not deterministic, contrary to λ_{fire} (the proof that normalization and strong normalization coincide makes sense only for λ_{sh}). Moreover here, unlike [5], we investigate in detail the case of terms reducing to values and how the general results for λ_{sh} can be applied to analyze qualitative and quantitative properties of Plotkin's λ_v restricted to closed terms (see above).

Recently, Mazza, Pellissier and Vial [37] introduced a general, elegant and abstract framework for building intersection (idempotent and non-idempotent) type systems characterizing normalization in different calculi. However, such a work contains a wrong claim in one of its applications to concrete calculi and type systems, confirmed by a personal communication with the authors: they affirm that the same type system as the one used here characterizes normalization in Plotkin's λ_v (endowed with the reduction $\to_{\beta_v^\flat}$), but we have shown on pp. 64–65 that this is false for open terms. Indeed, the property called full expansiveness in [37] (which entails that "normalizable $\Rightarrow$ typable") actually does not hold in λ_v. It is still true that their approach can be applied to characterize termination in Plotkin's λ_v restricted to closed terms and in the shuffling calculus λ_{sh}. Proving that the abstract properties described in [37] to characterize normalization hold in closed λ_v or in λ_{sh} amounts essentially to show that subject reduction (our Prop. 8), subject expansion (our Prop. 10) and typability of normal forms (our Lemma 15) hold.

The shuffling calculus λ_{sh} is compatible with Girard's call-by-value translation of λ-terms into linear logic (LL) proof-nets: according to that, λ-values (which are the only duplicable and erasable λ-terms) are the only λ-terms translated as boxes; also, sh-reduction corresponds to cut-elimination and $\mathsf{sh}^\flat$-reduction corresponds to cut-elimination at depth 0 (i.e. outside exponential boxes). The exact correspondence has many technical intricacies, which are outside the scope of this paper, anyway it can be recovered by composing the translation of the value substitution calculus (another extension of Plotkin's λ_v) into LL proof-nets (see [1]), and the encoding (studied in [4]) of λ_{sh} into the value substitution calculus. The relational semantics studied here is nothing but the relational semantics for LL (see [17]) restricted to fragment of LL that is the image of Girard's call-by-value translation. The notion of "experiment" in [17] corresponds to our type derivation, and the "result" of an experiment there corresponds to the conclusion of a type derivation here. The main results of de Carvalho, Pagani and Tortora de Falco [17] are similar to ours: characterization of normalization for LL proof-nets, extraction of quantitative information from (results of) experiments. Nonetheless, the properties shown here for λ_{sh} cannot be derived by simply analyzing the analogous results for LL proof-nets (proven in [17]) within its call-by-value fragment. Indeed, Ex. 28 shows that some property, which holds in the — apparently — more general case of untyped LL proof-nets (as proven in [17]), does not hold in the — apparently — special case of terms in λ_{sh}. It could seem surprising but, actually, there is no contradiction because LL proof-nets in [17] always require an explicit constructor for dereliction, whereas λ_{sh} is outside of this fragment since variables correspond in LL proof-nets to exponential axioms (which keep implicit the dereliction).

All the papers cited in this section are (more or less explicitly) inspired by de Carvalho's seminal work [15, 16], which first used relational semantics and non-idempotent intersection types to count the number of β-steps to reach the normal form in the call-by-name λ-calculus. Our results, although analogous and proven following an approach similar to [15, 16], cannot be derived directly from [15, 16]: indeed, the call-by-name λ-calculus corresponds to a different fragment of LL than call-by-value (as said in Sect. 1, call-by-name and call-by-value λ-calculi are translated into LL via two distinct embeddings). There is also another difference: de Carvalho [15, 16] counts the number of β-steps in *linear* call-by-name evaluation, which substitutes the argument of a β-redex for one variable occurrence at a time; here we compute the number of β-steps in *non-linear* call-by-value evaluation, which substitutes the argument of a β_v-redex for all the free occurrences of the redex-variable in just one step. A comprehensive study of the quantitative information given by non-idempotent intersection type systems for several (linear and non-linear) variants of call-by-name evaluation is provided in [3]. In [6] it has been introduced a non-idempotent intersection type system that combines some features of both call-by-name and call-by-value systems, providing quantitative information about the number of β-steps to reach the normal form by call-by-need evaluation.

References

[1] Beniamino Accattoli (2015): *Proof nets and the call-by-value λ-calculus*. *Theor. Comput. Sci.* 606, pp. 2–24, doi:10.1016/j.tcs.2015.08.006.

[2] Beniamino Accattoli & Claudio Sacerdoti Coen (2015): *On the Relative Usefulness of Fireballs*. In: *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015)*, IEEE Computer Society, pp. 141–155, doi:10.1109/LICS.2015.23.

[3] Beniamino Accattoli, Stéphane Graham-Lengrand & Delia Kesner (2018): *Tight typings and split bounds*. *PACMPL* 2(ICFP), pp. 94:1–94:30, doi:10.1145/3236789.

[4] Beniamino Accattoli & Giulio Guerrieri (2016): *Open Call-by-Value*. In Atsushi Igarashi, editor: *Programming Languages and Systems - 14th Asian Symposium (APLAS 2016)*, *Lecture Notes in Computer Science* 10017, Springer, pp. 206–226, doi:10.1007/978-3-319-47958-3_12.

[5] Beniamino Accattoli & Giulio Guerrieri (2018): *Types of Fireballs*. In Sukyoung Ryu, editor: *Programming Languages and Systems - 16th Asian Symposium (APLAS 2018)*, 11275, Springer, pp. 45–66, doi:10.1007/978-3-030-02768-1_3.

[6] Beniamino Accattoli, Giulio Guerrieri & Maico Leberle (2019): *Types by Need*. In Luís Caires, editor: *Programming Languages and Systems - 28th European Symposium on Programming (ESOP 2019)*, *Lecture Notes in Computer Science* 11423, Springer, pp. 410–439, doi:10.1007/978-3-030-17184-1_15.

[7] Beniamino Accattoli & Luca Paolini (2012): *Call-by-Value Solvability, Revisited*. In Tom Schrijvers & Peter Thiemann, editors: *Functional and Logic Programming - 11th International Symposium (FLOPS 2012)*, *Lecture Notes in Computer Science* 7294, Springer, pp. 4–16, doi:10.1007/978-3-642-29822-6_4.

[8] Hendrik Pieter Barendregt (1984): *The Lambda Calculus – Its Syntax and Semantics*. *Studies in Logic and the Foundations of Mathematics* 103, North Holland, Amsterdam, doi:10.1016/B978-0-444-87508-2.50006-X.

[9] Erika De Benedetti & Simona Ronchi Della Rocca (2016): *A type assignment for λ-calculus complete both for FPTIME and strong normalization*. *Inf. Comput.* 248, pp. 195–214, doi:10.1016/j.ic.2015.12.012.

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

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

[12] Antonio Bucciarelli, Thomas Ehrhard & Giulio Manzonetto (2012): *A relational semantics for parallelism and non-determinism in a functional setting*. *Ann. Pure Appl. Logic* 163(7), pp. 918–934, doi:10.1016/j.apal.2011.09.008.

[13] Antonio Bucciarelli, Delia Kesner & Daniel Ventura (2017): *Non-idempotent intersection types for the Lambda-Calculus*. *Logic Journal of the IGPL* 25(4), pp. 431–464, doi:10.1093/jigpal/jzx018.

[14] Alberto Carraro & Giulio Guerrieri (2014): *A Semantical and Operational Account of Call-by-Value Solvability*. In Anca Muscholl, editor: *Foundations of Software Science and Computation Structures - 17th International Conference (FOSSACS 2014)*, *Lecture Notes in Computer Science* 8412, Springer, pp. 103–118, doi:10.1007/978-3-642-54830-7_7.

[15] Daniel de Carvalho (2007): *Sémantiques de la logique linéaire et temps de calcul*. Thèse de doctorat, Université Aix-Marseille II.

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

[17] Daniel de Carvalho, Michele Pagani & Lorenzo Tortora de Falco (2011): *A semantic measure of the execution time in linear logic*. *Theor. Comput. Sci.* 412(20), pp. 1884–1902, doi:10.1016/j.tcs.2010.12.017.

[18] Daniel de Carvalho & Lorenzo Tortora de Falco (2016): *A semantic account of strong normalization in linear logic*. *Inf. Comput.* 248, pp. 104–129, doi:10.1016/j.ic.2015.12.010.

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

[20] Mario Coppo & Mariangiola Dezani-Ciancaglini (1980): *An extension of the basic functionality theory for the λ-calculus*. *Notre Dame Journal of Formal Logic* 21(4), pp. 685–693, doi:10.1305/ndjfl/1093883253.

[21] Alejandro Díaz-Caro, Giulio Manzonetto & Michele Pagani (2013): *Call-by-Value Non-determinism in a Linear Logic Type Discipline*. In Sergei N. Artëmov & Anil Nerode, editors: *Logical Foundations of Computer Science, International Symposium (LFCS 2013)*, *Lecture Notes in Computer Science* 7734, Springer, pp. 164–178, doi:10.1007/978-3-642-35722-0_12.

[22] Thomas Ehrhard (2012): *Collapsing non-idempotent intersection types*. In Patrick Cégielski & Arnaud Durand, editors: *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference*, *LIPIcs* 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 259–273, doi:10.4230/LIPIcs.CSL.2012.259.

[23] Thomas Ehrhard & Giulio Guerrieri (2016): *The Bang Calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value*. In James Cheney & Germán Vidal, editors: *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming (PPDP 2016)*, ACM, pp. 174–187, doi:10.1145/2967973.2968608.

[24] Philippa Gardner (1994): *Discovering Needed Reductions Using Type Theory*. In: *Theoretical Aspects of Computer Software (TACS '94)*, *Lecture Notes in Computer Science* 789, Springer, pp. 555–574, doi:10.1007/3-540-57887-0_115.

[25] Jean-Yves Girard (1987): *Linear Logic*. *Theor. Comput. Sci.* 50, pp. 1–102, doi:10.1016/0304-3975(87)90045-4.

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

[27] Benjamin Grégoire & Xavier Leroy (2002): *A compiled implementation of strong reduction*. In Mitchell Wand & Simon L. Peyton Jones, editors: *Proceedings of the Seventh International Conference on Functional Programming (ICFP '02)*, ACM, pp. 235–246, doi:10.1145/581478.581501.

[28] Giulio Guerrieri (2015): *Head reduction and normalization in a call-by-value lambda-calculus*. In Yuki Chiba, Santiago Escobar, Naoki Nishida, David Sabel & Manfred Schmidt-Schauß, editors: *2nd International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE 2015)*, *OASICS* 46, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 3–17, doi:10.4230/OASIcs.WPTE.2015.3.

[29] Giulio Guerrieri (2018): *Towards a Semantic Measure of the Execution Time in Call-by-Value lambda-Calculus (Long Version)*. *CoRR* abs/1812.10799. Available at `http://arxiv.org/abs/1812.10799`.

[30] Giulio Guerrieri, Luca Paolini & Simona Ronchi Della Rocca (2015): *Standardization of a Call-By-Value Lambda-Calculus*. In Thorsten Altenkirch, editor: *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, *LIPIcs* 38, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 211–225, doi:10.4230/LIPIcs.TLCA.2015.211.

[31] Giulio Guerrieri, Luca Paolini & Simona Ronchi Della Rocca (2017): *Standardization and Conservativity of a Refined Call-by-Value lambda-Calculus*. *Logical Methods in Computer Science* 13(4), doi:10.23638/LMCS-13(4:29)2017.

[32] Neil D. Jones, Carsten K. Gomard & Peter Sestoft (1993): *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[33] Delia Kesner & Daniel Ventura (2015): *A Resource Aware Computational Interpretation for Herbelin's Syntax*. In Martin Leucker, Camilo Rueda & Frank D. Valencia, editors: *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium*, *Lecture Notes in Computer Science* 9399, Springer, pp. 388–403, doi:10.1007/978-3-319-25150-9_23.

[34] Delia Kesner & Pierre Vial (2017): *Types as Resources for Classical Natural Deduction*. In Dale Miller, editor: *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, *LIPIcs* 84, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 24:1–24:17, doi:10.4230/LIPIcs.FSCD.2017.24.

[35] Assaf J. Kfoury (2000): *A linearization of the Lambda-calculus and consequences*. *J. Log. Comput.* 10(3), pp. 411–436, doi:10.1093/logcom/10.3.411.

[36] Jean-Louis Krivine (1993): *Lambda-calculus, types and models*. Ellis Horwood series, Ellis Horwood, Upper Saddle River, NJ, USA.

[37] Damiano Mazza, Luc Pellissier & Pierre Vial (2018): *Polyadic approximations, fibrations and intersection types*. *PACMPL* 2(POPL), pp. 6:1–6:28, doi:10.1145/3158094.

[38] Peter Møller Neergaard & Harry G. Mairson (2004): *Types, potency, and idempotency: why nonlinearity and amnesia make a type system work*. In Chris Okasaki & Kathleen Fisher, editors: *Proceedings of the Ninth International Conference on Functional Programming (ICFP 2004)*, ACM, pp. 138–149, doi:10.1145/1016850.1016871.

[39] Luca Paolini (2001): *Call-by-Value Separability and Computability*. In Antonio Restivo, Simona Ronchi Della Rocca & Luca Roversi, editors: *Theoretical Computer Science, 7th Italian Conference (ICTCS 2001)*, *Lecture Notes in Computer Science* 2202, Springer, pp. 74–89, doi:10.1007/3-540-45446-2_5.

[40] Luca Paolini, Mauro Piccolo & Simona Ronchi Della Rocca (2017): *Essential and relational models*. *Mathematical Structures in Computer Science* 27(5), pp. 626–650, doi:10.1017/S0960129515000316.

[41] Gordon D. Plotkin (1975): *Call-by-Name, Call-by-Value and the lambda-Calculus*. *Theor. Comput. Sci.* 1(2), pp. 125–159, doi:10.1016/0304-3975(75)90017-1.

[42] Garrel Pottinger (1980): *A type assignment for the strongly normalizable λ-terms*. In J.R. Hindley J.P. Seldin, editor: *To HB Curry: essays on combinatory logic, λ-calculus and formalism*, Academic Press, pp. 561–577.

[43] Laurent Regnier (1992): *Lambda-calcul et réseaux*. PhD thesis, Univ. Paris VII.

[44] Laurent Regnier (1994): *Une équivalence sur les lambda-termes*. *Theor. Comput. Sci.* 126(2), pp. 281–292, doi:10.1016/0304-3975(94)90012-4.

[45] Simona Ronchi Della Rocca & Luca Paolini (2004): *The Parametric Lambda Calculus - A Metamodel for Computation*. Texts in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/978-3-662-10394-4.