Intersection Types and (Positive) Almost-Sure Termination

UGO DAL LAGO, Università di Bologna & INRIA
CLAUDIA FAGGIAN, Université de Paris, IRIF, CNRS, Paris, France
SIMONA RONCHI DELLA ROCCA, Università di Torino

Randomized higher-order computation can be seen as being captured by a $\lambda$-calculus endowed with a single algebraic operation, namely a construct for binary probabilistic choice. What matters about such computations is the probability of obtaining any given result, rather than the possibility or the necessity of obtaining it, like in (non)deterministic computation. Termination, arguably the simplest kind of reachability problem, can be spelled out in at least two ways, depending on whether it talks about the probability of convergence or about the expected evaluation time, the second one providing a stronger guarantee. In this paper, we show that intersection types are capable of precisely characterizing both notions of termination inside a single system of types: the probability of convergence of any $\lambda$-term can be underapproximated by its type, while the underlying derivation's weight gives a lower bound to the term's expected number of steps to normal form. Noticeably, both approximations are tight—not only soundness but also completeness holds. The crucial ingredient is non-idempotency, without which it would be impossible to reason on the expected number of reduction steps which are necessary to completely evaluate any term. Besides, the kind of approximation we obtain is proved to be optimal recursion theoretically: no recursively enumerable formal system can do better than that.

1 INTRODUCTION

The study and analysis of randomised computation is almost as old as theoretical computer science itself [27, 60, 62]. In randomised computation, algorithms may well violate determinism by performing some inherently stochastic operations, like the one consisting in triggering probabilistic choice. In the last fifty years, randomised computation has been shown to enable efficient algorithms [51], but also secure cryptographic primitives (e.g. public-key cryptosystems [37]), which are provably impossible to define in a purely deterministic computational model.

Research on programming languages featuring various forms of random choice operators has itself a long history [49, 61], but has shown a strong impetus in the last ten years, due to progress in so-called bayesian programming languages [38, 64], in which not only probabilistic choice is available, but also conditioning has a counterpart inside programs, usually in the form of observe or score statements. In an higher-order scenario, the mere presence of a probabilistic choice operator, however, poses a number of challenges to the underlying theory. For example, relational reasoning by way of systems of logical relations [9], or by way of coinduction [24] has proved to be possible, although requiring some new ideas, both definitionally, or in the underlying correctness proof. Moreover, giving a satisfactory denotational semantics to higher-order languages with binary probabilistic choice is notoriously hard [41, 42], and has been solved in a completely satisfactory way only relatively recently [31, 39].

Types and Verification. Verification of deterministic higher-order programs can be carried out in many ways, including model checking [55], abstract interpretation [20], and type systems [56]. Among the properties one is interested in verifying programs against, safety and reachability are arguably the simplest ones. Type systems, traditionally conceived as lightweight methodologies ensuring safety (hence the slogan “well-typed programs cannot go wrong”), can also be employed to...
check reachability and termination [40, 63]. This idea has been brought to its extreme consequences by the line of work on intersection types [15, 18], which not only guarantee termination, but also characterise it, this way providing a compositional presentation of all and only the terminating programs. Indeed, intersection types can be seen as giving semantics to higher-order programs [6], and also to support program verification in subrecursive languages [46].

Probabilistic termination double nature. But what it means for a probabilistic program to terminate or—slightly more generally—to reach a state in which certain conditions hold? A first answer consists in considering a program terminating if the probability of divergence is null, namely if the program is almost surely terminating (AST for short). This way, even when the possibility of diverging is still there, it has null probability. This, however, does not mean that the time to termination (better, the expected time to termination) is finite: this is a stronger and computationally more meaningful requirement, called positive almost sure termination (shortened to PAST in the following). It is in fact well-known that checking programs for (positive) almost-sure termination turns out to be strictly harder, recursion theoretically, than checking termination of deterministic programs [43]: both almost-sure termination and positive almost-sure termination are not recursively enumerable, and have incomparable recursion-theoretic statuses, the former being \( \Pi^0_2 \)-complete, the latter being \( \Sigma^0_2 \)-complete. The discrepancy with the realm of deterministic calculi can be seen also in sub-universal languages: recently, Kobayashi, Dal Lago and Grellois [47], have shown that model checking reachability properties is undecidable in probabilistic higher-order recursion schemes, while the same problem is well known to be decidable in their deterministic and nondeterministic siblings [55]. More generally, the nature of probabilistic termination in presence of higher types is still not completely understood, and is fundamentally different from the one of its deterministic counterpart.

Some Natural Questions. Given the rich theory that the programming language community has been able to build for the deterministic \( \lambda \)-calculus, a number of questions naturally arise. Is it possible to faithfully and precisely reflect the expected time to termination by a system of types? What are the limits to the expressive power of such a system, given the aforementioned recursion theoretic limitations? Do intersection types can be of help, given their successes in characterising various notions of termination in a deterministic setting? These questions are natural ones, but have remained unanswered so far. This paper is the first one giving answers to them.

Contributions. We show here that intersection types indeed capture both forms of probabilistic termination in untyped probabilistic \( \lambda \)-calculi. More specifically, we define a system of non-idempotent intersection types such that from any type derivation for a given term \( M \), one can extract (in an effective, and even efficient, way) both a lower bound to the expected time to termination for \( M \), and a lower bound to \( M \)'s probability of termination. Remarkably, both kinds of bounds are tight, i.e. for every \( \epsilon > 0 \) there is a type derivation for \( M \) which gives an \( \epsilon \)-precise bound to both the probability of and the expected time to termination. The main novelty of the proposed methodology is the presence of distinct ingredients within the same type system, namely monadic types [23], intersection types [15], and non-idempotency [26]. Their contemporary presence forces us to switch from a purely qualitative notion of intersection (i.e. multisets) to a quantitative one (i.e. scaled multisets). This is necessary to appropriately deal with the multiple uses of program variables in presence of probabilistic choices. In view of the non-recursive enumerability of either kinds of probabilistic termination, taking type derivations as approximate witnesses to termination, rather than proper ones, indeed makes sense, and is the best one can do: we prove that any (recursively

\footnote{The term was introduced in [10], but the requirement that the program be expected to terminate is natural and fundamental, and was already present in [61]}

, Vol. 1, No. 1, Article . Publication date: October 2020.
A GENTLE INTRODUCTION TO INTERSECTION TYPES, TERMINATION, AND RANDOMIZATION

This section is meant to introduce the non-specialist to intersection types seen as a characterisation of terminating deterministic programs, and to the challenges one faces when trying to generalise intersection types to calculi featuring binary probabilistic choice.

2.1 Intersection Types and Termination

Suppose we work within a simple functional programming language, expressed as a call-by-value (CbV) $\mu$-calculus $\Lambda^{cbv}$ in A-normal form. Values and terms are generated through the grammars:

Values, $V^{cbv}$:

$$V ::= x \mid \lambda x.M$$

Terms, $\Lambda^{cbv}$:

$$M ::= V \mid VM \mid let M = x in M$$

Evaluation of closed terms is captured by two reduction rules, namely $$(\lambda x.M)V \rightarrow M\{V/x\}$$ and $let V = x in M \rightarrow M\{V/x\}$, which can be applied in any evaluation contexts, i.e. in any expression from the grammar $E ::= [] \mid let E = x in M$. As customary when working with functional languages, evaluation is weak (i.e., no reduction can take place in the body of a $\lambda$-abstraction).

This language can be seen as a fragment of Plotkin’s CbV $\mu$-calculus [57] in which the latter can be faithfully embedded. As such, the calculus is easily seen to be Turing-universal, and termination is thus an undecidable—although recursively enumerable—problem. How could we compositionally guarantee termination of those $\mu$-terms? The classic answer to the question above consists in endowing the calculus with a system of types. As an example, a system of simple types for the terms in $\Lambda^{cbv}$ is in Figure 1, where types are either an atom $\alpha$ or an arrow type $A \rightarrow B$.

Compositionality

### Fig. 1. Simple Types

| A ::= $\alpha \mid A \rightarrow A$ |
|----------------------------------|
| $\Gamma, x : A \vdash x : A$       |
| $\Gamma \vdash V : A \rightarrow B$ |
| $\Gamma \vdash W : A$              |
| $\Gamma \vdash VW : B$             |
| $\Gamma, x : A \vdash M : B$       |
| $\Gamma, x : A \vdash M : B$       |
| $\Gamma \vdash \text{let } N = x \text{ in } M : B$ |

Simple types indeed show that typability ensures termination. The converse does not hold, i.e. simple types are highly incomplete as a way to type terminating terms. As an example, self-application, namely the value $\lambda x.xx$, is not simply-typable even if terminating, since the variable $x$ cannot be assigned both the type $A$ and the type $A \rightarrow B$.

One way to go towards a type system complete for termination consists in resorting to some form of polymorphism. For example, parametric polymorphism in the style of System $F$ [36] dramatically increases the expressive power of simple types by way of a form of (second-order) quantification: the type $\forall \alpha.A$ stands for all types which can be obtained as formal instances of $A$. Parametric

2 an application $MN$ becomes the term $let M = x in let N = y in xy$
polymorphism, however, is not enough to get to a complete system, which can instead be built around ad-hoc polymorphism: rather than extending simple types by way of quantifiers, one can enrich types with intersections in the form of finite sets of types $\mathcal{A} = \{A_1, \cdots, A_n\}$, and take arrow types as expressions in the form $\mathcal{A} \rightarrow \mathcal{B}$. The type $\mathcal{A}$ can be assigned to terms which have type $\lambda x. M$ for every $j \in \{1, \ldots, n\}$. The resulting type system is in Figure 2, and is well-known to be both sound and complete for termination.

There is even more. One can make type derivations capable of reflecting quantitative kinds of information such as the number of required evaluation steps, rather than merely termination (which is qualitative in nature). This requires taking intersection types not as sets, but rather as multisets, i.e. $\mathcal{A} = [A_1, \cdots, A_n]$. This form of intersection type is dubbed non-idempotent, due to the non-idempotency of multiset unions and intersections. Type environments need now be treated multiplicatively rather than additively, this way giving a linear flavour to the type system. In non-idempotent intersection types, a natural number $w$ can be assigned to any type derivation in such a way that $\frac{w}{\lambda x. M} : [\ ]$ (where $[\ ]$ is the empty multiset seen as an intersection type) if and only if $M$ can be reduced to normal form in exactly $w$ steps. The resulting system is in Figure 3, and is essentially the one from [2].

### 2.2 Typing Termination in a Probabilistic Setting

How about probabilistic $\lambda$-calculi? Can the story in Section 2.1 be somehow generalised to such calculi? Endowing the class of terms with an operator for fair\(^3\) binary probabilistic choice is relatively easy: the grammar of terms needs to be extended by way of the production $M := M \oplus M$, and the term $M \oplus N$ evolves to either $M$ or $N$ with probability $\frac{1}{2}$, turning reduction on terms from a deterministic transition system to a Markov Chain with countably many states. Let us illustrate all this by way of an example, which will be our running example throughout the paper.

**Example 1 (Running Example).** Let us consider the term $DD$ where $D = \lambda x. (xx \oplus I)$, and $I$ is the identity $\lambda y.y$. The program $DD$ reduces to $DD \oplus I$, which in turn reduces to either $DD$ or $I$ with

\(^3\)Accomodating an operator for general binary probabilistic choice (e.g. in the form $\oplus_q$, where $q$ is a rational between 0 and 1) would be harmless, but would result in heavier notation; we thus prefer to stick to the fair case.
equal probability $1/2$. It is easy to see that after $2n$ steps, $DD$ has terminated with probability $\sum_{1}^{n} \frac{1}{2^n}$: while running $DD$, only one among the $2^n$ possible outcomes of the $n$ coin-flips results in the term staying at $DD$, all the others leading to $I$. Noticeably, the expression above tends to $1$ when $n$ tends to infinity. By weighting the steps with their probability, we have that the expected number of steps for $DD$ to terminate is $4$. In other words $DD$ is not only almost surely terminating, but positively so.

As this example shows, despite the minimal changes to the underlying operational semantics, reasoning about randomised computations can be more intricate than in the usual deterministic setting. More specifically:

- **Output.** While a deterministic program maps inputs to outputs, a probabilistic program maps inputs to *distributions* of outputs. For example, $DD$ evaluates to the Dirac distribution where all the probability is concentrated in the term $I$. Notice that this level of certitude is reached only at the limit, not in any finite amount of steps.

- **Termination.** A deterministic program either terminates on a given input or not. As we mentioned in the Introduction, a probabilistic program may give rise to diverging runs, still being almost-surely terminating. This is precisely what happens when evaluating $DD$: there is one run, namely the one always staying at $DD$, which diverges, but this run has of course null probability.

- **Runtime.** If a deterministic program terminates, it reaches its final state in finitely many steps, and we interpret this number as the time to termination. In the probabilistic case, what interests us is rather the expected number of steps, that is the average number of steps of the program’s runs. Such expected value may or may not be finite, even in the case of AST programs. When evaluating $DD$, this number is finite, but it arises (once again) as the sum of an infinite numerical series.

Small variations on Example 1 are sufficient to obtain terms whose behavior is more complex than that of $DD$. The following example illustrate that a term $M$ can reach *countably* many distinct normal forms and intermediate values.

**Example 2.** Let $\overline{0}, \overline{1}, \ldots, \overline{n}$, be $\lambda$-terms which encode the natural numbers $0, 1, \ldots, n$, and let $SUCC$ and $EXP$ be terms which encode the successor and the exponential function, respectively.

- **Consider the term $CC$ where $C = \lambda x. (\mathsf{succ}(xx) \oplus \overline{0})$, and $\mathsf{succ}(M)$ is sugar for $(\text{let } z = M \text{ in } SUCC \, z)$. Note that $\mathsf{succ}(\overline{n})$ reduces to $\overline{n+1}$ in constant time $K$, which depends on the choosen encoding. The program $CC$ reduces—at the limit—to each natural number $n$ with probability $\frac{1}{2^n}$. It is clear that $CC$ is AST, and it is easy to check that it is PAST; indeed it is expected to terminate in $4 + K$ steps. However its reduction graph, contrarily to the one of $DD$, involves infinitely many normal forms.

- **The term $\exp(CC)$, where $\exp(M)$ is sugar for $(\text{let } z = M \text{ in } EXP \, z)$, is a term which is still almost surely terminating, but not positively. Indeed, its expected runtime is infinite.**

All this shows that typing probabilistically terminating programs requires us to go significantly beyond classic intersection type theories, but also beyond the few attempts on type theories for probabilistic $\lambda$-calculi in the literature.

Let us now take a look at how the term $DD$ could be given an intersection type, in a way reflecting its being (positively) almost surely terminating. Let us write $DD$ as $D_1 D_2$. The term $D_1$ uses its argument $D_2$ in two different ways, the first as a function and the second as an argument to the same function. We already know that intersection types are there precisely for this purpose. But there are some fundamental differences here compared to the deterministic case: first of all, the two copies of $D_2$ that the function $D_1$ consumes are *used* only with probability $\frac{1}{2}$. Moreover, $D_1$ returns two different results, namely $I$ and $D_2 D_2$, each with equal probability. These two observations
inform how non-idempotent intersection types can be generalised to a \( \lambda \)-calculus with probabilistic choice. Indeed, the multisets \( \mathcal{A} \) and \( \mathcal{B} \) in an arrow type \( \mathcal{A} \rightarrow \mathcal{B} \) have to be enriched with some quantitative information:

- in order to capture the termination probability, the intersection type \( \mathcal{B} \) needs to be turned into a distribution of intersection types, reflecting the fact that the output of a computation is not one single value, but rather a distribution of them.
- capturing time expectations requires typing to become even more sophisticated, introducing two novelties:
  - The multiset of types \( \mathcal{A} \) needs to carry some information about the probability of each copy of the argument to be actually used. In other words, elements of \( \mathcal{A} \) needs to be scaled. Note the discrepancy between the ways \( \mathcal{A} \) and \( \mathcal{B} \) are treated: in the former a form of scaled multiset suffices, while in the latter a distribution of intersection types is needed.
  - Moreover, the type system needs to be capable of dubbing divergent terms as having arbitrarily large evaluation time expectations. Consider, as an example, the program \( M = I \oplus \Delta \Delta \), where \( \Delta = \lambda x.xx \). In one evaluation step, such a term reduces to the value \( I \) with probability \( \frac{1}{2} \) or to the diverging term \( \Delta \Delta \), with equal probability \( \frac{1}{2} \). The expected runtime of \( M \) is therefore infinite: \( 1 + \sum_{i=1}^{\infty} \frac{1}{2^i} \). Since typing \( M \) requires giving a type to \( \Delta \Delta \), the latter has to be attributed arbitrary large weights, although the only type it can receive is for obvious reasons the empty distribution.

We come back to all this in Section 5, after formally introducing the type system.

The aforementioned ones are not the only novelties of the type system we introduce in this paper. Given the already mentioned results by Kaminski et al. on the hardness of probabilistic termination [43], in which both notions of termination are proved not to be recursively enumerable, there is simply no hope to obtain results like the classic ones on deterministic terms, in which correctness of one derivation can serve as a termination certificate (this, to be fair, if checking type derivations for correctness remains decidable). The way out consists in looking at a characterisation by way of approximations: a type derivation would not be a witness of (positive) almost-sure termination by itself, but a witness of some lower-bound on the probability of termination or on the expected number of steps to termination. The type system needs to be tailored for this purpose.

3 A PROBABILISTIC CALL-BY-VALUE \( \lambda \)-CALCULUS

In this section, we formally introduce the minimalistic probabilistic functional programming language we have sketched in Section 2.1, and that we indicate in the following as \( \Lambda_{\text{cbv}}^{\oplus} \). We start with some technical definitions, which we will use throughout the paper.

3.1 Mathematical Preliminaries

Multisets. We denote a finite multiset (over a set \( \mathcal{X} \)) as \( [a_j \mid j \in J] \), where the index set \( J \) is finite and possibly empty. The empty multiset is denoted as \([\,]\), while elements of a non-empty multiset are often enumerated, like in \([a, b, c]\). Multiset union is noted \( \oplus \).

Distributions. Let \( \Omega \) be a countable set. A function \( \mu : \Omega \rightarrow [0, 1] \) is a probability subdistribution if its norm \( \| \mu \| := \sum_{\omega \in \Omega} \mu(\omega) \) is less or equal to 1. It is a distribution if \( \| \mu \| = 1 \). Subdistributions are the standard way to deal with possibly diverging probabilistic computations. We write \( D(\Omega) \) for the set of subdistributions on \( \Omega \), equipped with the standard pointwise partial order relation: \( \mu \preceq \rho \) if \( \mu(\omega) \leq \rho(\omega) \) for each \( \omega \in \Omega \). The support of \( \mu \) is the set \( \{ \omega \mid \mu(\omega) > 0 \} \).

Multidistributions. Suppose \( \mathcal{X} \) is a countable set and let \( \mathbf{m} \) be a finite multiset of pairs of the form \( pM \), with \( p \in (0, 1] \), and \( M \in \mathcal{X} \). Then \( \mathbf{m} = [p_iM_i]_{i \in I} \) is said to be a multidistribution.
on \( \mathbb{X} \) if \( ||m|| := \sum_{i \in I} p_i \leq 1 \). For multidistributions, we use the notation \( m = \langle p_i M_i \rangle_{i \in I} \). The empty multidistribution is denoted \( \emptyset \) (note that \( ||\emptyset|| = 0 \)). We denote by \( M(\mathbb{X}) \) the set of all multidistributions on \( \mathbb{X} \). We indicate the multidistribution \( \langle M \rangle \) simply as \( M \). The (disjoint) sum of multidistributions is denoted as \( + \), and is a partial operation. The product \( q \cdot m \) of a scalar \( q < 1 \) and a multidistribution \( m \) is defined pointwise: \( q \cdot \langle p_1 M_1, \ldots, p_n M_n \rangle = \langle (q p_1) M_1, \ldots, (q p_n) M_n \rangle \). Intuitively, a multidistribution \( m \) is an intensional representation of a probability distribution: multidistributions do not satisfy the equation \( m = p \cdot m + (1 - p) \cdot \emptyset \). This being said, every multidistribution can be made to collapse to a distribution, by taking the sum of all of its elements referring to the same \( M \in \mathbb{X} \).

### 3.2 The Language \( \Lambda_{cbv}^{\text{cbv}} \)

This section is devoted to introducing the language. Values and terms are defined by the grammar

\[
V ::= x \mid \lambda x. M \\
M ::= V \mid V V \mid M + M \mid \text{let } M = x \text{ in } M
\]

where \( x \) ranges over a countable set of variables. \( \Lambda_{cbv}^{\text{cbv}} \) and \( V_{cbv}^{\text{cbv}} \) denote respectively the set of terms and of values. Free and bound variables are defined as usual. \( M\{N/x\} \) denotes the term obtained from the capture-avoiding substitution of \( N \) for all the free occurrence of \( x \) in \( M \). As usual, a program is a closed term. Through the paper we frequently use the following terms as examples:

\[
I := \lambda x. x \\
\Lambda := \lambda x. xx \\
D := \lambda x. (xx \oplus I)
\]

The program \( \Delta \Delta \) is the paradigmatic diverging term, while \( DD \) is our running example.

### 3.3 The Operational Semantics

The operational semantics of \( \Lambda_{cbv}^{\text{cbv}} \) is formalized through the notion of multidistribution as introduced in Section 3.1, following [5]. To understand why this is a convenient way to describe the probabilistic dynamics of programs, let us consider how terms in \( \Lambda_{cbv}^{\text{cbv}} \) could be evaluated.

The intended dynamics of the term \( M \oplus N \) is that it reduces to either \( M \) or \( N \), with equal probability \( \frac{1}{2} \). That is, the state of the program after one reduction step is \( M \) with probability \( \frac{1}{2} \) and \( N \) with probability \( \frac{1}{2} \). Consider, as an example, the term \( (I \oplus (II)) \oplus II \). Its evaluation is graphically represented in Figure 4a. The first computation step consists in performing a probabilistic choice,

![Diagram of evaluation](image-url)

Fig. 4

proceeding as \( I \oplus (II) \) or as \( II \) according to its outcome. While the latter branch ends up in \( I \) (which is a value) in one deterministic step, the former branch proceeds with another probabilistic choice, which results in either \( I \) or \( II \). Finally, after another reduction step, \( II \) is reduced to the identity. To the right of the reduction tree in Figure 4a, one can see, for each time step, a summary of the “status”
of all probabilistic branches, each paired with its probability. After three steps, all branches reduce to $I$, and indeed the probability of observing $I$ when reducing the term is altogether $\frac{1}{4} + \frac{1}{4} + \frac{1}{2} = 1$. A more interesting example is in Figure 4b, and consists in the evaluation of our running example $DD$.

All this can be conveniently formalised by means of multidistributions; each element corresponds to a branch, i.e., to a possible reduction path of the underlying program — a multidistribution is essentially a distribution on such paths\(^4\). If switching to distributions, we would loose the precise correspondence with probabilistic branches, since many branches are collapsed into one. This is the ultimate reason why we adopt multidistributions, and will be discussed further in Section 8.2.

Let $M(\Lambda^{cbv}_\oplus)$ denote the set of multidistributions on (closed) terms. We define a reduction relation $\Rightarrow \subseteq M(\Lambda^{cbv}_\oplus) \times M(\Lambda^{cbv}_\oplus)$, given in Figure 5 and Figure 6, respectively. More precisely, we proceed as follows:

- We first define a reduction relation $\to$ from terms to multidistributions, e.g., $M \oplus N \to \langle \frac{1}{2}M, \frac{1}{2}N \rangle$. The one-step reduction $\Rightarrow \subseteq \Lambda^{cbv}_\oplus \times M(\Lambda^{cbv}_\oplus)$ is defined in Figure 5. A term $M$ is normal, or in normal form, if there is no $m$ such that $M \to m$. Please notice that closed terms are in normal form precisely when they are values. Finally, $\Rightarrow$ is deterministic, i.e., for every term $M$ there is at most one $m$ such that $M \to m$.

- Then we lift reduction of terms to reduction of multidistributions in the natural way, obtaining $\Rightarrow \models \text{e.g., } \langle \frac{1}{2}I, \frac{1}{2}(M \oplus N) \rangle \Rightarrow \langle \frac{1}{2}I, \frac{1}{2}M, \frac{1}{2}N \rangle$. The relation $\Rightarrow \subseteq \Lambda^{cbv}_\oplus \times M(\Lambda^{cbv}_\oplus)$ lifts to a relation $\Rightarrow \subseteq M(\Lambda^{cbv}_\oplus) \times M(\Lambda^{cbv}_\oplus)$ as defined in Figure 6. The way $\Rightarrow$ is defined implies that all reducible terms in the underlying multidistributions are actually reduced according to $\to$.

**Reduction sequences.** A $\Rightarrow$-sequence (or reduction sequence) from $m$ is a sequence $m = m_0, m_1, m_2, \ldots$ such that $m_i \Rightarrow m_{i+1}$ for every $i$. Notice that since multidistribution reduction is deterministic, each $m_0$ has a unique maximal reduction sequence, which is infinite and which we write $\{m_n\}_{n \in \mathbb{N}}$. We write $m_0 \Rightarrow^\ast m$ to indicate the existence of a finite reduction sequence from $m_0$, and $m_0 \Rightarrow^k m$ to specify the number $k$ of $\Rightarrow$-steps. Given a term $M$ and $m_0 = \langle M \rangle$, the sequence $m_0 \Rightarrow m_1 \Rightarrow \ldots$ naturally models the evaluation of $M$; each $m_k$ expresses the “expected” state of the system after $k$ steps.

\(^4\)In the spirit Markov Decision Processes, see e.g. [59].
Example 3. The term DD from Example 1 evaluates as follows:

\[ \langle DD \rangle \Rightarrow \langle DD \oplus I \rangle \Rightarrow \left( \frac{1}{2} DD, \frac{1}{2} I \right) \]

\[ \Rightarrow \left( \frac{1}{2} DD \oplus I, \frac{1}{2} I \right) \Rightarrow \left( \frac{1}{4} DD, \frac{1}{4} I, \frac{1}{2} I \right) \]

\[ \Rightarrow \left( \frac{1}{4} DD \oplus I, \frac{1}{4} I, \frac{1}{2} I \right) \Rightarrow \left( \frac{1}{8} DD, \frac{1}{8} I, \frac{1}{4} I, \frac{1}{2} I \right) \Rightarrow ... \]

The first three reduction steps match precisely what we have informally seen in Figure 4b.

Example 4. The term CC of Example 2 (where \( C = (\lambda x. \text{succ}(xx) \oplus \bar{0}) \)), evaluates as follows:

\[ \langle CC \rangle \Rightarrow \langle \text{succ}(CC) \oplus \bar{0} \rangle \Rightarrow \left( \frac{1}{2} \text{succ}(CC), \frac{1}{2} \bar{0} \right) \]

\[ \Rightarrow \left( \frac{1}{2} \text{succ}(\text{succ}(CC) \oplus \bar{0}), \frac{1}{2} \bar{0} \right) \Rightarrow \left( \frac{1}{4} \text{succ}(\text{succ}(CC)), \frac{1}{4} \text{succ}(\bar{0}), \frac{1}{2} \bar{0} \right) \]

where then

- \( \left\langle \frac{1}{2^n} \text{succ}^n(CC) \right\rangle \Rightarrow \left\langle \frac{1}{2^n} \text{succ}^n(\text{succ}(CC) \oplus \bar{0}) \right\rangle \Rightarrow \left\langle \frac{1}{2^n} \text{succ}^n(\text{succ}(CC)), \frac{1}{2^n} \text{succ}^n(\bar{0}) \right\rangle \ldots \)
- \( \text{succ}^n(\bar{0}) \Rightarrow n^K \bar{n} \).

Observe that the evaluation of CC is similar to that of DD. However, while in Example 3 the term I is a value, \( \text{succ}^n(\bar{0}) \) is not. It still has to perform \( n^K \) steps (where \( K \) is a constant, given in Example 2) in order to reduce to the value \( \bar{n} \).

Values and Multidistributions. Given a multidistribution \( m \in M(\Lambda_{\oplus}) \), we indicate by \( m^V \) its restriction to values. Hence if \( m = \langle p_i | M_i \rangle \) \(_{i \in I} \), then \( ||m^V|| := \sum_{M_i \in V} p_i \). The real number \( ||m^V|| \) is thus the probability that \( m \) is a value, and we will refer to it this way. Looking at Example 3, observe that after, e.g., four reduction steps, DD becomes the value I with probability \( \frac{1}{2} + \frac{1}{4} \). More generally, after \( 2n \) steps, DD is a value with probability \( \sum_{k=1}^{n} \frac{1}{2^k} \).

3.4 Probabilistic Termination in \( \Lambda_{\oplus}^{cbv} \)

Let \( M \) be a closed term, and \( \langle M \rangle = m_0 \Rightarrow m_1 \Rightarrow m_2 \ldots \) the reduction sequence which models its evaluation. We write \( \text{PTerm}_k(M) \) for \( ||m_k^V|| \), which expresses the probability that \( M \) terminates in at most \( k \) steps.

Termination. Given a closed term \( M \), the probability of termination of \( M \) is easily defined by \( \text{PTerm}(M) := \sup_n \left\{ \text{PTerm}_n(M) \right\} \). As an example \( \text{PTerm}(DD) \) is \( \sup_n \sum_{k=1}^{n} \frac{1}{2^k} = \sum_{k=1}^{\infty} \frac{1}{2^k} = 1 \).

Expected Runtime. We now define the expected runtime of \( M \), following the literature [5, 33, 43]. As pointed out in [33] the expected runtime can be expressed\(^5\) in a very convenient form as, informally,

\[ \text{ETime}(M) = \sum_{k \geq 0} \text{Pr}["M runs more than } k \text{ steps"]} \]

\[ = \sum_{k \geq 0} (1 - \text{Pr}["M terminates within } k \text{ steps"]}) \].

\(^5\)This because the runtime is a random variable taking values into \( \mathbb{N} \); we therefore can easily compute its expectation by using the telescope formula, see e.g. [11], page 27. The equivalence with the formulations we give below is spelled out in [5].
Within our setting, the above is easily formalised as follows.

\[ \text{ETIME}(M) = \sum_{k \geq 0} (1 - \text{PTerm}_k(M)) \]

This formulation can be given a very intuitive interpretation: given the reduction sequence \( \langle M \rangle = m_0 \Rightarrow m_1 \Rightarrow ...m_k \Rightarrow ... \) each tick in time (i.e. each \( \Rightarrow \) step) is weighted with its probability to take place —more precisely, the probability that a redex is fired. Since only (and all) terms which are not in normal form reduce, the system in state \( m_i \) reduces with probability \( 1 - ||m_i^V|| \).

**Finite Approximants.** Given a term \( M \), the number \( \text{PTerm}_n(M) \) is a finite approximant (the \( n \)-th approximant) of \( \text{PTerm}(M) \). It is useful to define finite approximants for \( \text{ETIME}(M) \) too:

\[ \text{ETIME}_n(M) := \sum_{k=0}^{n-1} (1 - \text{PTerm}_k(M)) \]

Clearly \( \text{ETIME}(M) = \sup_n \{ \text{ETIME}_n(M) \} \).

**Example 5 (Expected runtime, and its approximants).** Consider again the evaluation of the term \( DD \); let us decorate each step \( m_k \Rightarrow m_{k+1} \) with the expected probability that a redex is actually fired in \( m_k \), that is \( 1 - ||m_k^V|| \):

\[
\langle DD \rangle \xrightarrow{1} \langle DD \oplus I \rangle \xrightarrow{\frac{1}{2}} \left\langle \frac{1}{2} DD, \frac{1}{2} I \right\rangle \xrightarrow{\frac{1}{2}} \left\langle \frac{1}{2} DD \oplus I, \frac{1}{2} I \right\rangle \xrightarrow{\frac{1}{2}} \left\langle \frac{1}{4} DD, \frac{1}{4} I, \frac{1}{4} I \right\rangle \xrightarrow{\frac{1}{8}} \left\langle \frac{1}{8} DD, \frac{1}{8} I, \frac{1}{8} I \right\rangle \xrightarrow{\frac{1}{8}} ... \]

It is immediate to verify that \( \text{ETIME}(M) = 4 \). As for the approximants, we have that, e.g. \( \text{ETIME}_2(M) = 2 \), \( \text{ETIME}_4(M) = 3 \), \( \text{ETIME}_6(M) = 3 + \frac{1}{2} \).

**(Positive) Almost Sure Termination in \( \Lambda^{\text{cbv}} \).** We now have all the ingredients to define the two concepts this paper aims to characterise, namely the two canonical notions of termination. The definition turns out to be very easy.

**Definition 6.** Let \( M \) be a closed term. We then say that \( M \) is **almost surely terminating** (AST) if \( \text{PTerm}(M) = 1 \). Furthermore, we say that \( M \) is **expected to terminate**, or **positively almost surely terminating** (PAST) if \( \text{ETIME}(M) \) is finite.

As is well-known, PAST is strictly stronger than AST. Indeed, it is easily seen that PAST implies AST:

**Fact 7.** For every closed term \( M \), \( \text{ETIME}(M) < \infty \) implies that \( \text{PTerm}(M) = 1 \). Indeed, \( \sum_{i \geq 1} (1 - \text{PTerm}_i(M)) < \infty \) implies \( \lim_{i \to \infty} (1 - \text{PTerm}_i(M)) = 0 \), hence \( \lim_{i \to \infty} \text{PTerm}_i(M) = 1 \).

However, a program may be AST, and still have infinite expected runtime. The paradigmatic example of this is a fair random walk [8]. With a slight abuse of notation, we often use AST and PAST both as acronyms and as sets of terms.

## 4 NON-IDEMPOTENT MONADIC INTERSECTION TYPES

In the previous section, we have introduced a call-by-value paradigmatic programming language for probabilistic computation, endowed it with an operational semantics, and defined two notions of probabilistic termination for it. In this section, we present a type system which is able to capture both the **probability of termination** and the **expected runtime** of a program. The system will in turn allow us to characterise both AST and PAST.
Intersection Types and (Positive) Almost-Sure Termination

One of the main ingredients of the type system we are going to introduce is the non-idempotency of intersections. As sketched in Section 2.1, such a type system is usually based on two mutually recursive syntactic categories of types, namely simple (or arrow) types and intersection types, which are finite multisets of arrows. The intuition is that an arrow type corresponds to a single use of a term, and that if an argument is typed with a multiset containing \( k \) arrows types, it will be evaluated \( k \) times.

In our probabilistic setting, the type system is based on three, rather than two, layers, namely arrow types, intersection types, and multidistribution types, also known as monadic types. More precisely:

- An **arrow type** corresponds to a single use of a value, as usual. In a purely applicative language like ours, indeed, the only way to destruct a value is to pass another value to it.
- An **intersection type**, instead, is no longer a multiset of arrows like in usual non-idempotent intersection type disciplines, but a multiset of pairs \( q_A \), where \( A \) is an arrow type, and \( q \in (0,1) \cap Q \). The intuition is that each single use of a term will happen with some probability \( q \), and that \( q \) is recorded in the intersection type together with the corresponding arrow. So, e.g., in the evaluation of \( DD \) (see Example 3) the argument \( D \) is first used with certitude (probability 1); its next use happens with probability \( \frac{1}{2} \), the following use with probability \( \frac{1}{4} \), and so on. Each use is typed with an (appropriately scaled) arrow.
- Finally, a term \( M \) cannot in general be typed “with certitude” namely by a single intersection type \( A \), but rather with a **multidistribution** of intersection types \( \langle p_1A_1, \ldots, p_kA_k \rangle \). Indeed, the evaluation of \( M \) can result in possibly many values depending on the probabilistic choices the term encounters along the way. In turn, those values can be copied, and each possible use of them must be taken into account.

We now formally introduce the type system. In Section 5 we expand the intuitions above by analysing some type derivations of our main example \( DD \). To understand the typing, the reader should not hesitate to jump back and forth between the examples and the formal system.

### 4.1 The Type System, Formally

**Types.** Types are defined by means of the following grammar:

\[
\begin{align*}
\mathcal{A}, \mathcal{B} & ::= \mathcal{A} \rightarrow a \\
\mathcal{A}, \mathcal{B} & ::= \{q_1A_1, \ldots, q_nA_n\}, n \geq 0 \\
a, b & ::= \langle p_1A_1, \ldots, p_nA_n \rangle, n \geq 0
\end{align*}
\]

In other words, an intersection type \( \mathcal{A} \) is a multiset of pairs \( q_A \) where \( A \) is an arrow type, and \( q \in (0,1) \cap Q \) is said to be a scale factor. Note that \( q > 0 \). Letters \( u, q \) range over scale factors. Given \( \mathcal{A} = [q_iA_i]_{i \in I} \), we write \( u.\mathcal{A} \) for \( [(uq_i).A_i]_{i \in I} \).

It is useful to notice that an intersection type is not a multidistribution, because the sum of the \( q_i \) such that \( \mathcal{A} = [q_iA_i]_{i \in I} \) is not bounded by 1 in general; this is reflected in distinct bracket notations. Intersection types and type distributions are indeed fundamentally different. Each element in an intersection type corresponds to one use of the term; so for example, \( AA \) can have a type of the form \( [1.A, 1. [A \rightarrow a] \). Instead, type distributions are probabilistic sums of possibly different intersection types. We often need to multiply intersection types or type distributions by scalars, getting other objects of the same kind. Moreover, intersection types and type distributions being multisets, they support the (respective) operation of disjoint union (see Section 3.1).

**Contexts.** A typing context \( \Gamma \) is a (total) map from variables to intersection types such that only finitely many variables are not mapped to the empty multiset \([\]\). The domain of \( \Gamma \) is the

}. Vol. 1, No. 1, Article . Publication date: October 2020.
This section provides some explanation on the shapes and roles of the typing rules. The leaves of type derivations seen as trees can be of two kinds, namely the \(\lambda\)-rule and the \(\text{let}\) rule. In both cases the underlying weight is set to 0. While the former is standard in intersection type disciplines, the latter attributes the empty distribution 0 to any term \(M\). So for example, \(\Delta\Delta\) is typed as \(0\). The purpose of \(\text{ZERO}\) is to allow for approximations, by allowing the typing process to stop at any point.

The next four rules are concerned with the four term constructors \(\Lambda_{\Box}^{\text{cbv}}\) includes, namely applications, probabilistic sums, abstractions, and \(\text{let}\)s. The following discusses each of them:

- The \(\lambda\) rule types a lambda abstraction, and assigns an arrow type to it. This poses no problem, because the type assigned to the variable \(x\) in the underlying typing context is an intersection type, and this matches the shape of the left-hand-side of an arrow type. The weight is increased by one: whenever this abstraction will be used as a function, a \(\beta\)-redex would fire, and this takes one reduction step, which needs to be counted.
- The \(\oplus\) rule types \(M \oplus N\) by “superimposing” the derivations for \(M\) and for \(N\). The data carried by each such derivation (context, weight and type) are scaled by a factor of \(\frac{1}{2}\). The counter is increased by 1, to record a \(\oplus\)-step in the evaluation.
- The \(\text{let}\) rule serves to type the \(\text{let}\) construct, and is probably the most complex one. In particular, the argument \(M\) needs to be typed multiple types, one for each scaled multiset in...
the multidistribution \(<p_k \mathcal{A}_k\rangle\), which is the type for the first argument \(N\). Each subderivation will be used with probability \(p_k\), therefore in the conclusion of the \(let\)-rule, the data of each of the leftmost premisses (typing context \(\Delta_k\), weight \(w_k\), and type \(b_k\)) are scaled by a factor \(p_k\). Moreover, the weight is further increased by 1, to record a \(letV\)-step in the evaluation; such a step consumes the \(let\) when the first argument is a value.

- Finally, the \(@\) rule typing applications is quite standard in shape. Just a couple of observations could be helpful. First of all, the function \(V\) is required to be typed with a multiset, rather than a distribution, and this is not restrictive since \(V\) is a value, and not a term. Secondly, the weight is taken as the sum of the weights of the two derivations, without any increase. Notice that the corresponding \(\beta\)-step is recorded by the \(\lambda\)-rule.

The last two rules, namely \(\text{Val}\) and \(!\), are the only ones not associated to any term construction, and are meant to allow a term typable with arrow types to be attributed an intersection or distribution type. Of course, this makes sense only when the term is actually a value.

Remark 8. While designing the type system, we made a simplifying choice in the typing rule \(let\). As we said, the counter is increased by 1 to record the \(letV\)-step. Note however that if \(N\) is a non-terminating term, it never becomes a value, and therefore the \(letV\)-step never happens. Are we counting too much here? Obviously not, because if \(N\) never become a value, then any reduction sequence from \(let M = x \in N\) can be extended with an extra reduction step, without affecting the analysis in any way.

Some Interesting Boundary Cases. The type system we have just introduced is remarkably simple in structure, despite its expressive power, which we will analyze in Section 6. Let us now take a look at a few degenerate cases of the typing rules:

1. In the \(\lambda\) rule, \(A\) is allowed to be the empty intersection type, this way allowing to type vacuous abstractions, i.e., we can always abstract a variable \(x\) which does not explicitly occur in the context \(\Gamma\), since if \(x \notin \text{dom}(\Gamma)\), then \(\Gamma, x : []\) is equal to \(\Gamma\).
2. In the \(!\)-rule, \(I\) can be empty, and the following rule is thus a derived rule:

\[
\begin{array}{c}
\vdash \lambda x.M : [] \\
\end{array}
\]

3. In the \(let\) rule, the term \(N\) can well have null type \(0\), and in this case the whole term \(let N = x \in M\) is given itself type \(0\), without any need to type \(M\). In other words, the following is another derived rule:

\[
\begin{array}{c}
\Gamma \vdash N : 0 \\
\Gamma \vdash let N = x \in M : 0 \\
\end{array}
\]

4.3 Some Basic Properties of the Type System

In this section, we derive some easy but useful properties of the type system, which will turn out to be essential in the following. Like in linear type systems, typing contexts tell us everything we need to know about free variables:

Lemma 9 (Contexts and Free Variables). Let \(\Gamma \vdash M : a\). Then \(\text{dom}(\Gamma) \subseteq \text{fv}(M)\), and \(M\) closed implies \(\Gamma = \emptyset\).

The way intersection types are assigned to values is completely determined by the underlying arrow types:

Property 10 (Partitioning Intersections). For every value \(V\), the following are equivalent:
(1) \( \vdash V : \mathcal{A} \) and \( \mathcal{A} = \psi[\mathcal{A}]_{i} \).
(2) \( \vdash V : \mathcal{A}_{i} \) for every \( i \in I \) and \( w = \sum w_{i} \).

We often use the aforementioned property together with the following lemma:

**Lemma 11 (Scaling).** Given any scalar \( 0 < q \leq 1 \) and any value \( V \), it holds that \( q^{\alpha} \vdash V : q.\mathcal{A} \) iff \( \vdash V : \mathcal{A} \).

## 5 Precisely Reflecting the Runtime: Some Examples

Our type system is designed to keep track of the probability of reaching a value and the expected time to termination. As and it should by now be clear, the information relevant to derive the latter is kept track by the weight. Since the expected runtime is computed as an infinitary sum, working with exact measures is essential. Think for example at \( I \) and the only difficulty consists in distinguishing between those abstractions which are used as functions, and those abstractions which will turn out to be the final value.

In a probabilistic setting, we want to compute the expected runtime. Increasing the weight by one at each instance of the \( \lambda \) rule as in the deterministic case is simply too much. Consider our running example \( DD \), where \( D = \lambda x.xx \odot I \), and \( \text{ETime}(DD) = 4 \). It is easy to see that for each \( k \in \mathbb{N} \), there is a derivation \( \Pi \) which contains \( k \) instances of the \( \lambda \) rule. If each is counted \( \lambda \)-abstraction in the type derivation corresponds a \( \beta \)-redex being fired, therefore the first converges, while the second diverges.

The only difficulty consists in distinguishing between those abstractions which are used as functions, and those abstractions which will turn out to be the final value. The only difficulty consists in distinguishing between those abstractions which are used as functions, and those abstractions which will turn out to be the final value. In this section, we illustrate these aspects by way of some concrete examples.

### 5.1 Not Too Much

In intersection type systems for the \( \lambda \)-calculus such as those by Lengrand and co-authors [1, 3, 7], the weight of any type derivation accounts for how many times redexes can be fired. Roughly speaking, to each \( \lambda \)-abstraction in the type derivation corresponds a \( \beta \)-redex being fired, therefore to measure the runtime of a \( \lambda \)-term, the weight is increased by one at each instance of the \( \lambda \) rule. The only difficulty consists in distinguishing between those abstractions which are used as functions, and those abstractions which will turn out to be the final value.

In a probabilistic setting, we want to compute the expected runtime. Increasing the weight by one at each instance of the \( \lambda \) rule as in the deterministic case is simply too much. Consider our running example \( DD \), where \( D = \lambda x.xx \odot I \), and \( \text{ETime}(DD) = 4 \). It is easy to see that for each \( k \in \mathbb{N} \), there is a derivation \( \Pi \) which contains \( k \) instances of the \( \lambda \) rule. If each is counted \( \lambda \)-abstraction in the type derivation corresponds a \( \beta \)-redex being fired, therefore

\[ \text{ETime}(DD) = 4 \]

The fact that \( \mathbb{P} \text{Term}_{2n}(DD) \) is \( \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2n} \). We define the types \( \mathcal{A}_{n} \) and \( \mathcal{A}_{n} \) as follows:

\[ \mathcal{A}_{0} = [] \quad \mathcal{A}_{n} = \mathcal{A}_{n-1} \rightarrow \left[ \frac{1}{2^{k}} \right] \quad \mathcal{A}_{n} = \frac{1}{2} \cdot \mathcal{A}_{n-1} \odot \frac{1}{2} \cdot \mathcal{A}_{n} \]

For the reader’s convenience, we explicitly give some cases:

\[ \mathcal{A}_{1} = [] \rightarrow \left[ \frac{1}{2} \right] \], \quad \mathcal{A}_{2} = \mathcal{A}_{1} \rightarrow \left[ \frac{1}{4} \right], \quad \mathcal{A}_{3} = \mathcal{A}_{2} \rightarrow \left[ \frac{1}{8} \right] \]

\[ \mathcal{A}_{1} = \left[ \frac{1}{2} \cdot \mathcal{A}_{1} \right], \quad \mathcal{A}_{2} = \left[ \frac{1}{4} \cdot \mathcal{A}_{1}, \frac{1}{2} \cdot \mathcal{A}_{2} \right], \quad \mathcal{A}_{3} = \left[ \frac{1}{8} \cdot \mathcal{A}_{1}, \frac{1}{4} \cdot \mathcal{A}_{2}, \frac{1}{2} \cdot \mathcal{A}_{3} \right] \]
The value $D$ can be given all the arrow types $\lambda_i$, for every $i$, all these derivations having weight equal to 2, i.e. for every $i \geq 1$ there is a derivation $\Sigma_i$ such that $\Sigma_i \vdash 2 \lambda x. xx \oplus I : \lambda_i$. For example, the type derivations $\Sigma_1$ to $\Sigma_3$ can be built as follows:

\[
\begin{align*}
0 & \vdash I : \langle [] \rangle & x : [\lambda_i] \vdash x : [\langle \frac{1}{2} [i] \rangle] & x : [\langle \frac{3}{2} \rangle] x : [] \vdash x : [] \\
1 & \vdash xx \oplus I : \langle \frac{3}{2} [i] \rangle & x : [\langle \frac{1}{2} \rangle] \vdash xx : \langle \frac{3}{2} [i] \rangle & \vdash I : \langle [] \rangle \\
\Sigma_1 \vdash 2 \lambda x. xx \oplus I : [] \to \langle \frac{3}{2} [i] \rangle & \Sigma_2 \vdash 2 \lambda x. xx \oplus I : [\frac{1}{2}, \lambda_i] \to \langle \frac{1}{2}, \frac{1}{2} [i] \rangle \\
\end{align*}
\]

Collecting together the $\Sigma_i$’s by way of the rule $!$, one can also attach the intersection type $\mathcal{A}_i$ to $D$, this time with weight $2(\Sigma_{k:1}^{i} \frac{1}{2^k}) = 2 - \frac{1}{2^i}$ thus obtaining the type derivation $\Theta_i$. Finally, $\Sigma_{i+1}$ and $\Theta_i$ can be aggregated in the derivation $\Phi_{i+1}$, typing $DD$. Note that the weight is now $2(\Sigma_{k:0}^{i} \frac{1}{2^k})$.

\[
\begin{align*}
\Sigma_{i+1} \vdash 2 \vdash D : \mathcal{A}_i \to \langle [i]^{+1} [k:1] \rangle & \Theta_i \vdash 2(\Sigma_{k:1}^{i} \frac{1}{2^k}) D : \mathcal{A}_i \\
\Phi_{i+1} \vdash 2(\Sigma_{k:0}^{i} \frac{1}{2^k}) DD : \langle [i]^{+1} [k:1] \rangle \\
\end{align*}
\]

For example, we have the following derivation, which indeed corresponds to the $(2 \cdot 3)$-approximant of $\text{ETime}(DD)$: recall from Example 5 that $\text{ETime}_6(DD) = 3 + \frac{1}{2}$ (and that $\text{PTime}_6(DD) = \frac{1}{2}$).

\[
\begin{align*}
\Sigma_1 \vdash 2 \vdash D : \lambda_i & \Sigma_2 \vdash 2 \vdash D : \lambda_i \\
\Sigma_3 \vdash 2 \lambda x. xx \oplus I : [\frac{1}{2}, \lambda_i, \lambda_i] \to \langle \frac{1}{3} [i], \frac{1}{4} [i], \frac{1}{2} [i] \rangle & \vdash 14 \frac{1}{2} D : [\frac{1}{2}, \lambda_i, \lambda_i] \\
\Phi_3 \vdash 3^{+\frac{1}{2}} DD : \langle \frac{1}{8} [i], \frac{1}{4} [i], \frac{1}{2} [i] \rangle \\
\end{align*}
\]

### 5.2 Not Too Little

Our type system allows to count the reduction steps of diverging terms. That is, a term such as $\Delta \Delta$ has a derivation of weight $n$, for each $n \in \mathbb{N}$. This is essential to precisely capture the expected runtime. Think of the term $M := I \oplus \Delta \Delta$. Its evaluation proceeds as follows:

\[
\langle I \oplus \Delta \Delta \rangle \Rightarrow \langle \frac{1}{2}, \frac{1}{2} \Delta \Delta \rangle \Rightarrow \langle \frac{1}{2}, \frac{1}{2} \Delta \Delta \rangle \Rightarrow \langle \frac{1}{2}, \frac{1}{2} \Delta \Delta \rangle \Rightarrow \langle \frac{1}{2}, \frac{1}{2} \Delta \Delta \rangle \ldots
\]

Clearly, $\text{ETime}(M) = \infty$. However, any derivation only taking into account the evaluation time to a value (namely the $\oplus$ reduction step only), would necessarily have finite weight. In the following, we prove that any diverging program $M$ can be typed as $\vdash M : 0$, for every natural number $n$. 
Here, we show this fact, concretely, for the paradigmatic diverging term $\Delta\Delta$. First of all, consider the arrow types $A_i + 1 = [A_1, \ldots, A_i] \to 0$ (so, in particular, $A_1 = [] \to 0$, $A_2 = [A_1] \to 0$). For each $i$, one can build a derivation $\Sigma_i$ having weight 1 and typing $\Delta$ with $A_i$. Here are a couple of examples:

\[
\begin{align*}
\Sigma_1 & \vdash \lambda x. x : [][\to 0] \to 0 & \Sigma_2 & \vdash \lambda x. x : [[] \to 0] \to 0 \\
0 & \vdash xx : 0 & 0 & \vdash xx : 0 
\end{align*}
\]

From the $\Sigma_i$’s, it is thus easy to build derivations typing $\Delta\Delta$ with 0 and having any weight $n$. As an example, if $n = 3$, we have the following one:

\[
\begin{align*}
\Sigma_3 & \vdash \lambda x.xx : [[][\to 0] \to 0, [][\to 0] \to 0 & \Sigma_2 & \vdash \lambda x.xx : [[] \to 0] \to 0 & \Sigma_1 & \vdash \lambda x.xx : [] \to 0 \\
& & & & \vdash (\lambda x.xx)(\lambda x.xx) : 0 \quad \@ 
\end{align*}
\]

6 CHARACTERIZING PROBABILISTIC TERMINATION

This section presents the main result of this paper, namely the characterisation of both forms of probabilistic termination by typing. This will be done by relating type derivations for a program $M$ with the probability of termination and the expected runtime of $M$. To achieve the latter, we need to focus on tight derivations, since not all type derivations of $M$ underapproximate the expected runtime of $M$.

We show that in the tight case, $\text{ETime}(M)$ (respectively, $\text{PTerm}(M)$) bounds from above the weight $w$ (respectively, the norm $\|a\|$) of any type derivation $\Pi \vdash \vdash M : a$. This is the soundness property, and is in Section 6.2. We also prove the converse, i.e. the completeness property, in Section 6.3

6.1 Tight Typings

The need for tight typings can be grasped easily by considering the following example

**Example 12.** The term $I$ is in normal form, and therefore $\text{ETime}(I) = 0$. It can be given the type 0 (by way of the ZERO typing rule), or the type $\langle 1 \rangle$ (by way of $!$ and Val). In both cases, the underlying weight is 0. However, $I$ also admits derivations whose weight is strictly positive, such as

\[
\begin{align*}
\vdash x : [][\to 0] & \vdash x : [][\to 0] & \vdash \lambda x.x : [][\to 0] \\
\vdash VAR & & \lambda 
\end{align*}
\]

More generally, without any restrictions on the shape of types, one can easily assign grossly overapproximated weights to terms, e.g., the term $\lambda x.(\Delta\Delta)$, which is a value but which can receive arbitrarily large weights when given the type $[] \to 0$, (immediate consequence of the example in Section 5.2 above).

The purpose of arrow types is to give types to terms which are not supposed to be reduced alone, but only when applied to an argument. If, instead, a term is not supposed to be used as a function, its type must be the empty multiset. This is the key idea for understanding the following definition:

**Definition 13** (Tight Types and Derivations). A type $a$ is said to be tight if it is a multidistribution on the empty intersection type $[]$. Accordingly, a derivation $\Pi \vdash \vdash M : a$ is said to be tight if $a$ is tight.
A tight type has therefore shape \( a = \langle q_k \rangle_{k \in K} \), where \( K \) is possibly nonempty. In particular the null type \( 0 \) is a tight type. Observe that if \( a \) is tight, then \( \| a \| = \sum_k q_k \) (which, again, is null when \( K \) is empty). The following can be proved by quickly inspecting the typing rules:

**Lemma 14 (Tight Typings for Values).** If \( V \) is a closed value, then there are precisely two tight derivations for \( V \), both of weight 0:

\[
\frac{}{0 \vdash V : []} ! \quad \frac{}{0 \vdash V : \langle 1[\rangle}}[val]
\]

Looking back at Example 12, one immediately realizes that tightness allows us to get rid of overapproximations, at least for values. Does this lift to all terms? The next two subsections will give a positive answer to this question. The following property, which is immediate from the definitions, will be useful in the rest of this section.

**Property 15.** For any closed term \( M \) and any \( k \in \mathbb{N} \), it holds that \( \text{PTerm}_k(M) \leq \text{PTerm}_{k+1}(M) \) and \( \text{ETime}_k(M) \leq \text{ETime}_{k+1}(M) \). Moreover, if \( M \rightarrow \langle q_i M_i \rangle_{i \in I} \) then

\[
\text{PTerm}_{k+1}(M) = \sum_{i \in I} q_i(\text{PTerm}_k(M_i)), \quad \text{ETime}_{k+1}(M) = 1 + \sum_{i \in I} q_i(\text{ETime}_k(M_i)).
\]

### 6.2 Soundness

In this section, we prove the correctness of our type system. Namely, we prove that if \( \vdash M : a \) is (tightly) derivable, then \( M \) has probability of termination at least \( \| a \| \), and expected runtime at least \( w \).

The proof of correctness is based on the following, namely a form of weighted subject reduction, that for good reasons has a probabilistic flavor here. The size of a type derivation \( \Pi \) (denoted \( |\Pi| \)) is the standard one, and is defined as the number of rules in \( \Pi \), (excluding the \(!\)-rule and the \( \text{VAL}\)-rule, which cannot be iterated.)

**Lemma 16 (Weighted Subject Reduction).** Suppose that \( \Pi \vdash \vdash M : b \), with \( w > 0 \), and that \( P \rightarrow \langle q_i P_i \rangle_{i \in I} \). Then for every \( i \in I \) there exists a derivation \( \Pi_i \) such that \( \Pi_i \vdash \vdash P_i : b_i \), and \( |\Pi| > |\Pi_i| \). Moreover, \( b = \big|\big|_{i \in I} q_i|b_i| \) and \( w = 1 + \sum_{i \in I} q_i|w_i| \).

The proof is in the extended version. Notice how the type stays the same, at least on the average, while the weight strictly decreases. This in turn implies that whenever a term is (tightly) typable, its weight is a lower bound to its expected time to termination, while the norm of its type is a lower bound to the probability of termination. This is proved by way of approximations, as follows:

**Theorem 17 (Finitary Soundness ).** Let \( M \) be a closed term. For each tight typing \( \vdash M : b \), there exists \( k \in \mathbb{N} \) such that \( \| b \| \leq \text{PTerm}_k(M) \) and \( w \leq \text{ETime}_k(M) \).

**Proof.** By induction on the size \( |\Pi| \) of the type derivation \( \Pi \) such that \( \Pi \vdash \vdash M \), distinguishing some cases. Recall that for closed terms, the normal forms are exactly the values.

- If \( M \) is a value, the claim holds by Lemma 14, where we observe that \( w = 0 \). Notice that \( \text{PTerm}_0(M) = \| M \|^V \| = 1 \) and \( \text{ETime}_0(M) = 0 \).
- Otherwise, if \( M \) is not a value, we further distinguish some cases:
  - If \( w = 0 \), then by inspecting the rules, we see that the only derivable tight judgment is \( \vdash M : 0 \) which trivially satisfies the claim, with \( k = 0 \).
This section is devoted to characterisation results relating typing and termination. The latter can be given in three different ways, and we devote a subsection to each of them.

6.3 Completeness

The last section showed that type derivations provide lower bounds on the probability of convergence, and on the expected time to termination. It is now time to prove that tight derivations approximate with arbitrary precision the aforementioned quantities. The proof of completeness is based on the following probabilistic adaptation of Weighted Subject Expansion, whose proof is in the extended version.

**Lemma 19 (Weighted Subject Expansion).** Let \( P \) be a closed term. Assume that \( P \rightarrow \langle q_i P_i \rangle_{i \in \mathcal{I}} \) and that for each \( i \in \mathcal{I}, \Pi_i \vdash \frac{w_i}{w_i} P_i : a_i \). Then, there exists a single derivation \( \Pi \vdash \frac{w}{w} P : a \) such that \( a = \frac{w}{w} (q_ia_i) \) and \( w \geq 1 + \sum q_iw_i \).

We can now thus prove the dual to Theorem 17 above:

**Theorem 20 (Finitary Completeness).** Let \( M \) be a closed term. For each \( k \in \mathbb{N} \) there exists a tight derivation \( \Pi \vdash \frac{w}{w} M : a \), such that \( \Vert a \Vert = \text{PTerm}_k(M) \) and \( w \geq \text{ETime}_k(M) \).

**Proof.** By induction on \( k \), distinguishing some cases.

- If \( M \) is a value, then for each \( k \), \( \text{PTerm}_k(M) = 1 \) and \( \text{ETime}_k(M) = 0 \). The derivation \( \vdash \frac{0}{0} M : \langle 1 \rangle \) (Lemma 14) satisfies the claim.

- Otherwise, if \( M \) is not a value:
  - If \( k = 0 \), we have \( \text{PTerm}_0(M) = 0 \) and \( \text{ETime}_0(M) \); the Zero-rule satisfies the claim.
  - If \( k > 0 \), assume \( M \rightarrow \langle p_i M_i \rangle_{i \in \mathcal{I}} \). By \( i.h. \), for each \( i \in \mathcal{I}, \Pi_i \vdash \frac{w_i}{w_i} M_i : a_i \), such that \( \Vert a_i \Vert = \text{PTerm}_{k-1}(M_i) \) and \( w_i \geq \text{ETime}_{k-1}(M_i) \). By Weighted Subject Expansion, there exists a tight derivation \( \Phi \vdash \frac{w}{w} M : a \) such that \( \Vert a \Vert = \sum p_i \Vert a_i \Vert \), and \( w \geq \sum p_iw_i \). We conclude by Property 15, because \( \text{PTerm}_k(M) = \sum p_i \text{PTerm}_{k-1}(M_i) = \sum p_i \Vert a_i \Vert = \Vert a \Vert \) and \( \text{ETime}_k(M) = 1 + \sum p_i \text{ETime}_{k-1}(M_i) \leq 1 + \sum p_iw_i = w \).

\[ \square \]

6.4 The Various Flavours of a Correspondence

This section is devoted to characterisation results relating typing and termination. The latter can be given in three different ways, and we devote a subsection to each of them.
6.4.1 A Uniform Characterization. A characterization of both $\text{PTerm}(M)$ and $\text{ETime}(M)$ by the same class of derivations, namely tight derivations, can be given as follows:

**Theorem 21 (Tight Typing and Termination).** Let $M$ be a closed term. Then

$$\text{PTerm}(M) = \sup\{|a| \mid \exists \Pi. \Pi \vdash M : a \text{ is a tight derivation}\}$$

$$\text{ETime}(M) = \sup\{w \mid \exists \Pi, a. \Pi \vdash_w M : a \text{ is tight derivation}\}$$

**Proof.** Let us first of all define $N(M)$ and $W(M)$ as follows:

$$N(M) := \sup\{|a| \mid \Pi \vdash M : a \text{ is a tight derivation}\}$$

$$W(M) := \sup\{w \mid \Pi \vdash_w M : a \text{ is tight derivation}\}$$

We now proceed by proving the following two statements:

- On the one hand, we prove that $N(M) = \text{PTerm}(M)$, recalling that $\text{PTerm}(M) := \sup\{\text{PTerm}_n(M) \mid n \in \mathbb{N}\}$.
  - Let $\Phi \vdash \sup M : a$ be a tight derivation. By Correctness (Theorem 17), $|a| \leq \text{PTerm}(M)$. Hence $N(M) \leq \text{PTerm}(M)$.
  - By Finitary Completeness (Theorem 20), for each $k \in \mathbb{N}$ there exists a tight derivation $\Pi \vdash \sup M : a$ such that $\text{PTerm}_k(M) \leq |a|$. Hence $N(M) = \text{PTerm}(M)$.

- On the other hand, we prove that $W(M) = \text{ETime}(M)$, recalling that $\text{ETime}(M) := \sup\{\text{ETime}_n(M) \mid n \in \mathbb{N}\}$.
  - Let $w$ be the weight of a tight derivation $\Phi \vdash \sup M : a$. By Correctness (Theorem 17), $w \leq \text{ETime}(M)$. Hence $W(M) \leq \text{ETime}(M)$.
  - Again by Finitary Completeness (Theorem 20), for each $k \in \mathbb{N}$ there exists a tight derivation $\Pi \vdash \sup M : a$ such that $\text{ETime}_k(M) \leq w$. Hence $W(M) = \text{ETime}(M)$.

\[\square\]

By definition, $M$ is AST iff $\text{PTerm}(M) = 1$, while $M$ is PAST iff $\text{ETime}(M)$ is finite; therefore

**Corollary 22 (Tight Typing, AST, and PAST).** Let $M$ be a closed term. Then: $M$ is AST iff $\sup\{|a| \mid \exists \Pi. \Pi \vdash M : a \text{ is a tight derivation}\} = 1$. Moreover, $M$ is PAST iff $\sup\{w \mid \exists \Pi, a. \Pi \vdash_w M : a \text{ is tight derivation}\} < \infty$.

**Example 23.** In Section 5 we have discussed tight derivations for our running example $DD$. Each derivation $\Phi_j$ ($j \in \mathbb{N}$) has weight $W_j = 2 \sum_{n<j} \frac{1}{2^n}$, which is bounded by 4. It is interesting to briefly consider also type derivations for the terms $CC$ and $E$ of Example 2.

- For the term $CC$, we already noticed that its evaluation (Example 4) is similar to that of $DD$, but produces countably many different values. We can build a type derivation $\Phi_j^{CC}$ for $CC$ following the blueprint of the derivation $\Phi_j$ for $DD$. Now instead of the $j$ leaves of shape $\vdash^0 1 : \langle[]\rangle$ we would have the $j$ tight type derivations of $\text{succ}^n(\bar{0})$ ($0 \leq n < j$), each with respective weight $nk$. Taking into account scaling, the weight of $\Phi_j^{CC}$ is then $W_j + \frac{1}{2} K(\sum_{n<j} \frac{n}{2^n})$.

- A tight type derivation $\Phi_j^E$ for $E$ also can be built on the model of $\Phi$. Now the leaves are replaced by type derivations of terms $\exp(\text{succ}^n(\bar{0}))$, each with weight $\nu_n > \exp(n)$; each such $\nu_n$ is then scaled by $\frac{1}{2} \frac{1}{2^n}$. Since for each $j \in \mathbb{N}$, there is a tight type derivation whose weight exceeds $\sum_{n<j} \frac{1}{2^n} \exp(n)$, the set of weights is clearly unbounded.
6.4.2 Focusing on Expected Runtimes. Remember that PAST $\subseteq$ AST. The previous characterisation may give the impression that analysing the runtime of a term somehow requires studying its probability of termination. In fact, our type system allows us to establish PAST independently from AST, by looking only at the type $0$ rather than at all tight typings. The results we are going to prove tell us that if we are only interested in the expected runtime, we can indeed limit the search space to the derivations of the null type $0$. First of all, a strengthening of Finitary Completeness can be given

**Proposition 24 (Finitary Completeness of Null Typing).** Let $M$ be a closed term. Then, for each $k \in \mathbb{N}$ there exists a derivation $\Pi \vdash^w M : 0$, such that $w \geq ETime_k(M)$.

**Proof.** The proof is a simplification of Theorem 20. We only need to observe that the $0$ typing is preserved by subject expansion, and the weights strictly increases along it. \hfill \square

Since Finitary Soundness holds at all types, we can easily reach the following:

**Theorem 25 (Null Typing, Expected Runtimes, and PAST).** Let $M$ be a closed term. Then:

$$ETime(M) = \sup\{w \mid \Pi \vdash^w M : 0\} \quad M \in \text{PAST} \iff \sup\{w \mid \Pi \vdash^w M : 0\} < \infty$$

**The Running Example, Revisited.** Let us go back to our running example $DD$, and show that its runtime can be analysed by way of null types. We can indeed build type derivations of the form $\Pi \vdash^w DD : 0$ in such a way that $w$ is bounded by $ETime(DD) = 4$, and for each approximant $ETime_k(DD)$ there is a derivation which has at least that weight. The structure of these type derivations are identical to the ones we gave in Section 5. The only difference is in how the underlying types are defined. Let us define the families of types $\{B_n\}_{n \in \mathbb{N}}$ and $\{\bar{B}_n\}_{n \in \mathbb{N}}$ as follows:

$$B_0 = [\ ] \quad B_n = \frac{1}{2}B_{n-1} \cup \frac{1}{2}[B_n] \quad B_n = B_{n-1} \rightarrow 0$$

For example:

$$B_1 = [\ ] \rightarrow 0, \quad B_2 = B_1 \rightarrow 0, \quad B_3 = B_2 \rightarrow 0,$$

$$\bar{B}_1 = \left\{ \frac{1}{2}B_1 \right\} \quad \bar{B}_2 = \left\{ \frac{1}{4}B_1, \frac{1}{2}B_2 \right\}, \quad \bar{B}_3 = \left\{ \frac{1}{8}B_1, \frac{1}{4}B_2, \frac{1}{2}B_3 \right\}$$

The given types are structurally very similar to those from Section 5. We can thus mimic the constructions given there, and get derivations $\Sigma_i$, each having weight $2$ and typing $D$ with $B_i$, but also derivations having weights converging to $4$, this time typing $DD$ with $0$. So for example, recalling that $ETime_0(DD) = 3 + \frac{1}{2}$, here is the corresponding type derivation:

$$x : [[\frac{1}{2}B_1] \rightarrow 0] \vdash^0 x : [[\frac{1}{2}B_1] \rightarrow 0] \quad x : [\frac{1}{2}B_1] \vdash^0 x : [\frac{1}{2}B_1] \quad x : [\frac{1}{2}B_1] \vdash^0 x : [\frac{1}{2}B_1] \vdash^0 I : 0$$

$$\Sigma_3 \vdash^0 \lambda x.xx @ I : [\frac{1}{2}B_2, \frac{1}{2}B_1] \rightarrow 0$$

$$\Sigma_1 \vdash^0 D : B_1 \quad \Sigma_2 \vdash^0 D : B_2 \quad \vdash^0 D : [\frac{1}{2}B_2, \frac{1}{2}B_1]$$

$$\vdash^0 DD : 0$$
6.4.3 Focusing on the Probability of Termination. In this section, we have shown that our type system induces characterisations of both AST and PAST by the same family of derivations, namely the tight derivations. Moreover, we proved that we can restrict the search space to the class of null typings whenever interested in the expected number of steps, only. But there is more: if we are interested in the probability of termination only, an orthogonal simplification is possible—we could drop from the typing all the information on the scaling factors, as that is only used in deriving the weight.

7 ON RECURSION-THEORETIC OPTIMALITY

The uniform characterisation of both forms of termination we described in Section 6 is remarkable, because one single system is capable of providing precisely the kind of information one needs in either case:

- The (norm of the) underlying type is a lower (but tight) bound to the probability of termination.
- The weight of type derivations is a lower (but again tight) bound to the expected time to termination.

As usual in type systems, reasoning is compositional: the typings one attributes to composite terms are derived from those one assigns to the subterms. This being said, AST and PAST can only be verified at the limit, since all possible type derivations for the given term and having a certain conclusion need, in general, to be taken into account.

At this point, one may wonder whether one can do better than Theorem 21 when characterising probabilistic termination. Is it that one can get away from approximations, and devise a (possibly more complicated) type system in which one type derivation is by itself a certificate? In this section, we prove that under mild assumptions in fact one cannot, i.e. that our characterisation is the best possible, at least recursion-theoretically.

Our results are based on the well-known ones by Kaminski et al. [43], which establish that in the realm of probabilistic Turing machines, almost sure termination is a \( \Pi^0_2 \)-complete problem, while positive almost sure termination is \( \Sigma^0_2 \)-complete problem. We give two results in this section:

- On the one hand, we show that probabilistic Turing machines can be faithfully encoded into \( \Lambda_{cbv}^{\oplus} \), witnessing the fact that the aforementioned recursion-theoretic limitations also hold for \( \Lambda_{cbv}^{\oplus} \).
- On the other hand, we prove by way of our type system that the class of positively almost surely terminating terms in \( \Lambda_{cbv}^{\oplus} \) is \( \Sigma^0_2 \), which in view of the previous point means that our type system is as simple as possible, recursion-theoretically. A similar result is given for almost surely terminating terms and \( \Pi^0_2 \).

7.1 Probabilistic Turing Machines

Probabilistic Turing machines [35, 62] (PTMs in the following) can be defined similarly to ordinary deterministic ones, the main difference being the fact that the transition function \( \delta \) returns not one pair in \( \Sigma \times \{\leftarrow, \downarrow, \rightarrow\} \), but a distribution of those. Various restrictions might be imposed on the form of those distributions, without affecting the class of representable (random) functions, but only inducing some overhead. Here, we assume that the underlying distribution is a Bernoulli one, assigning probability \( \frac{1}{2} \) to one pair and probability \( \frac{1}{2} \) to another one. As usual, we can also assume to work with 1-tape Turing machines. Again, this is not restrictive. Both notions of termination we have introduced in Section 3.4 in the realm of \( \Lambda_{cbv}^{\oplus} \) make perfect sense for Turing machines too, e.g., given a probabilistic Turing machine \( M \) and an input \( x \in \Sigma^* \), we say that \( M \) is AST on \( x \) if \( M \) converges with probability 1. Like ordinary Turing machines, PTMs can be effectively enumerated and the PTM corresponding to \( \alpha \) is indicated as \( M_\alpha \). This allows us to introduce the following
classes of (pairs of) natural numbers:

\[
\text{AST}_{\text{TM}} = \{(\alpha, x) \mid \text{the PTM } \mathcal{M}_x \text{ is AST on input } x\}
\]

\[
\text{PAST}_{\text{TM}} = \{(\alpha, x) \mid \text{the PTM } \mathcal{M}_x \text{ is PAST on input } x\}
\]

7.2 Encoding PTMs into \(\Lambda^{\text{cbv}}\)

Let us now switch to the encoding of probabilistic Turing machines into \(\Lambda^{\text{cbv}}\). As a target language, we actually take a sub-class of terms of \(\Lambda^{\text{cbv}}\), namely the one defined by the following grammar:

\[
\begin{align*}
V & := x \mid \lambda x. M & \quad & \text{Values, } V^{\text{cps}} \\
M & := V \mid MV \mid M \oplus M & \quad & \text{Terms, } \Lambda^{\text{cps}}
\end{align*}
\]

where \(MV\) is nothing more than syntactic sugar for \(\text{let } M = x \text{ in } x V\). In doing so, we follow [21], and take as a target calculus for our encoding one in which only one redex is active in any term. This way, all our results will also be valid in Section 8.1, where intersection types will be generalised to a calculus with call-by-name evaluation.

The main ingredients of the encoding are the following ones:

- States and strings can be encoded following the so-called Scott scheme [67], e.g., given an alphabet \(\Sigma = \{a_1, \ldots, a_m\}\) strings in \(\Sigma^*\) are encoded following the recursive definition below:

  \[
  \bar{e} = \lambda x_1, \ldots, x_m. y. z \quad \bar{a}_i \bar{s} = \lambda x_1, \ldots, x_m. y. x_i \bar{s}
  \]

- Similarly, one can encode any tuple of values \((V_1, \ldots, V_n)\) as \(\lambda x. x V_1 \cdots V_n\). This encoding easily supports projections.

- We can build a fixed-point combinator \(Z\) as \(M M\), where \(M\) is the term \(\lambda x. \lambda y. y (\lambda z. x z y z)\). Observe that for every value \(V\), it holds that \(Z V\) deterministically rewrites (in a constant amount of steps) to \(V (\lambda x. Z V x)\). Notice that the argument to \(V\) is not \(Z V\), but is "wrapped" into a value by way of \(\eta\)-expansion: this is necessary, given the nature of our calculus.

Given the above, and after a fair amount of intermediate technical results (but closely following [21], except in the encoding of the transition function), one can reach the following:

**Theorem 26.** For every probabilistic Turing Machine \(M\), there is lambda term \(T_M\) such that the evaluation of \(T_M \bar{s}\) and the computation of \(M\) on input \(s\) produce the same distributions (up to encodings). Moreover, the number of steps taken by \(T_M\) is linearly related to \(M\). Finally, the term \(T_M\) can be effectively obtained from (the code of) \(M\).

7.3 Preliminaries from Recursion Theory

In this subsection, we give some basic definitions about the arithmetic hierarchy, for the sake of making this paper self-contained. An excellent reference about these topics is [54].

A set \(X \subseteq \mathbb{N}\) is said to be \(\Sigma^0_n\) iff there is a primitive recursive relation \(R \subseteq \mathbb{N}^{n+1}\) such that

\[
x \in X \iff \exists y_1. \forall y_2. \exists y_3. \forall y_4. \ldots R(x, y_1, \ldots, y_n)
\]

\(n\) times

Dually, \(X\) is said to be \(\Pi^0_n\) iff there is a primitive recursive relation \(R \subseteq \mathbb{N}^{n+1}\) such that

\[
x \in X \iff \forall y_1. \exists y_2. \forall y_3. \exists y_4. \ldots R(x, y_1, \ldots, y_n)
\]

\(n\) times

For both the classes \(\Sigma^0_n\) and \(\Pi^0_n\), there are related notions of hardness: a set \(X \subseteq \mathbb{N}\) is \(\Sigma^0_n\)-difficult (respectively, \(\Pi^0_n\)-difficult) iff it is at least as difficult as any other \(\Sigma^0_n\) (respectively, \(\Pi^0_n\)) problem, i.e. if for every other \(\Pi^0_n\) problem \(Y\) there is a (recursive) reduction from \(Y\) to \(X\). Both in \(\Sigma^0_n\) and in \(\Pi^0_n\),

\(\text{, Vol. 1, No. 1, Article . Publication date: October 2020.}\)
Intersection Types and (Positive) Almost-Sure Termination

Completeness stands for the containment and hardness. These classes form an hierarchy which is strict; moreover, $\Sigma^0_n$ and $\Pi^0_n$, although having non-empty intersections, are incomparable as sets.

Where, in the arithmetical hierarchy, do AST$_{TM}$ and PAST$_{TM}$ reside? A precise answer to this question has been given by Kaminski et al. [43] in the realm of while programs, but can easily be rephrased for PTMs:

**Theorem 27 (Kaminski et al. [43]).** AST$_{TM}$ is $\Pi^0_2$ complete, while PAST$_{TM}$ is $\Sigma^0_2$ complete.

Theorem 27 is quite surprising, in particular if seen through the lenses of ordinary, deterministic computation. In universal deterministic computational models (like TMs or the $\lambda$-calculus) terminating computations form a $\Sigma^0_1$-complete set: even if undecidable, the set is recursively enumerable, and any terminating computation can be endowed with a finite certificate, itself (effectively) checkable for correctness. This, by the way, is a recursive-theoretical justification of the possibility of building complete systems of intersection types for the deterministic $\lambda$-calculus in which type derivations play the role of certificates, as the ones we describe in Section 2: this is possible only because termination is in $\Sigma^0_1$.

### 7.4 The Optimality Result

In the probabilistic $\lambda$-calculus, neither form of termination is $\Sigma^0_1$, and as a consequence type derivations cannot play the role of certificates. In this section we will formally prove the statement above, along the lines showing that the form of approximation we employ is optimal.

First of all, we can give the $\lambda$-counterparts of AST$_{TM}$ and PAST$_{TM}$:

$$\text{AST}_\lambda = \{ M \mid M \text{ is AST} \} \quad \text{PAST}_\lambda = \{ M \mid M \text{ is PAST} \}$$

Theorem 26 and Theorem 27 together imply that AST$_\lambda$ is $\Pi^0_2$-hard and PAST$_\lambda$ is $\Sigma^0_2$-hard. But how about containment?

Actually, our characterisation results, namely Theorem 21 and Corollary 22 can be seen as a way to prove that AST$_\lambda$ is in $\Pi^0_2$ and that PAST$_\lambda$ is in $\Sigma^0_2$. Indeed, consider the following two sets

$$\text{AST}_{\lambda,\prec} = \{ M \mid \forall r \in Q_{(0,1)} . \exists \Pi . (\Pi \vdash w M : a) \land (|a| > r) \};$$

$$\text{PAST}_{\lambda,\prec} = \{ M \mid \exists r \in Q . \forall \Pi . (\Pi \vdash w M : a) \Rightarrow (w < r) \}.$$ 

By Corollary 22, AST$_{\lambda,\prec}$ = AST$_\lambda$ and PAST$_{\lambda,\prec}$ = PAST$_\lambda$. But by definition, AST$_{\lambda,\prec}$ is $\Pi^0_2$, because checking whether a natural number is the encoding of a type derivation $\Pi$ having the property that $(\Pi \vdash w M : a) \land |a| > r$ for given $M$ and $r$ is certainly a primitive recursive problem. Similarly for PAST$_{\lambda,\prec}$ and $\Sigma^0_2$.

This is why we claim that our intersection types are optimal: there cannot be simpler (in the sense of the arithmetical hierarchy) characterisations of the class of AST$_\lambda$ and PAST$_\lambda$.

### 8 Variations on the Theme

This section is devoted to analysing two variations on the type system we introduced in Section 4, itself proved to satisfy some nice properties, but certainly not being the only system of intersection types one can define in a discrete probabilistic setting.

#### 8.1 On Call-by-Name Evaluation

Despite the fact that the call-by-value discipline is more natural in presence of effects, it is legitimate to ask whether the system of intersection types we have designed can be adapted to CbN evaluation. This section is devoted to showing that this is actually the case.
As a language we use here the standard probabilistic untyped \(\lambda\)-calculus equipped with weak head reduction, itself already studied in many papers from the literature [24, 25]. We first define the language, called \(\Lambda_{cbn}^{\oplus}\), and its operational semantics, then the typing system.

The Language of Terms. Terms and values are defined by the grammar

\[
\begin{align*}
V &::= x \mid \lambda x.M \\
M &::= V \mid MM \mid M \oplus M
\end{align*}
\]

where \(x\) ranges over a countable set of variables. Observe how values are defined as in \(\Lambda_{cbv}^{\oplus}\), while terms are slightly different, and more in line with the usual \(\lambda\)-calculus. Another remark: the class \(\Lambda_{cbn}^{\oplus}\) is trivially a subclass of \(\Lambda_{cbv}^{\oplus}\).

Operational Semantics and Probabilistic Termination. As in CbV, we first define a one-step reduction relation \(\rightarrow\) from terms to multidistributions. The rules are given in Figure 8. We then lift \(\rightarrow\) to a reduction of multidistributions, and this can be done as for \(\Lambda_{cbv}^{\oplus}\), so following the rules in Figure 6. Values are precisely the closed terms which cannot be further reduced. The definitions of \(\mid m^n_0\|\), \(\text{PTerm}_k(M)\), \(\text{PTerm}(M)\), \(\text{ETime}_k(M)\), and \(\text{ETime}(M)\) can be given exactly as in Section 3.4. Again, observe how the semantics of all terms of \(\Lambda_{cbps}^{\oplus}\) is the same if defined through CbV, as we did originally, or through CbN, as we are doing here. As a consequence, all results from Section 7.2 also hold for CbN.

8.1.1 The Type System. Non-Idempotent Intersection types for the Call-by-Name \(\lambda\)-calculus [26, 34, 45, 52] are well-studied. We adapt them to our probabilistic setting. The types reflect the underlying semantics of all terms of \(\Lambda_{cbv}^{\oplus}\), and multidistribution types. Notice however that now a type distribution is a (multi)-distribution over arrows. An intersection type is a multiset of scaled types, i.e. a multiset of pairs \(q.a\) where \(a\) is a type distribution, and \(q \in (0, 1] \cap \mathbb{Q}\) is as usual a scale factor. Types are defined by means of the following grammar:

\[
\begin{align*}
\Lambda, \beta &::= * \mid \mathcal{A} \rightarrow a \\
\mathcal{A}, \beta &::= [q_1.a_1, \ldots, q_n.a_n] n \geq 0 \\
a, \beta &::= (p_1.A_1, \ldots, p_n.A_n) \ , n \geq 0
\end{align*}
\]

Observe the presence of the special arrow type \(*\), which here plays the role of the empty multiset \([\]\) in CbV.

Typing Rules. The type assignment system in Figure 7 proves judgements of the shape \(\Gamma \vdash M : T\), where \(\Gamma\) is a type context, \(M\) a term, \(w \in \mathbb{Q}\) is a counter, and \(T\) is either a or \(\mathcal{A}\). The notation \(q, \Gamma\) is as in Section 4.1, taking into account that now if \(\mathcal{A} = [q_1.a_1]_{i \in I}\), \(u.\mathcal{A}\) is \([u(q_i).a_i]_{i \in I}\). The notion of a tight type needs to be appropriately adapted:

Definition 28 (Tight Types and Derivations). A type \(a\) is said to be tight if it is a multidistribution on the arrow type \(*\). A derivation \(\Pi \vdash^w M : a\) is tight whenever \(a\) is tight.
Intersection Types and (Positive) Almost-Sure Termination

\( x : [\lambda a] \vdash x : a \)

\( \Gamma, x : A \vdash M : b \)

\( \Gamma \vdash \lambda x. M : (A \to b) \)

\( \Gamma \vdash \lambda x. M : (\langle A \to b \rangle)_{k \in K} \)

\( \Pi_k > \Delta_k \vdash N : \langle A \rangle_{k \in K} \)

\( \Gamma \vdash \Delta \vdash N : b \)

\( (\Gamma_i \vdash M : a)_{i \in I} \)

\( q_i \in I \)

\( \sum q_i \vdash M : [q_i.a_i] \quad (i \in I) \)

\( \frac{1}{2} \Gamma \vdash \frac{1}{2} \Delta \vdash M \oplus N : \frac{1}{2} a \oplus \frac{1}{2} b \)

Fig. 9. Non-Idempotent Intersection Type Rules for \( \Lambda_{\text{cbn}}^\oplus \).

Basic Properties. As in CbV, some basic properties of the type system are not only useful, but reveal the nature of the type system. First of all, any closed value \( V \) can be tightly typed with probability 1, by \( 0 \vdash V : \langle \ast \rangle \). Moreover, a degenerate form of the rule \( ! \) allows to derive the following for any term \( M \):

\( 0 \vdash M : [ ] \)

Finally, a useful instance of the \( @ \) rule is the following:

\( \Gamma \vdash M : 0 \)

\( \Gamma \vdash MN : 0 \)

8.1.2 Characterising CbN Probabilistic Termination. The just introduced type system allows us to transfer all results from Section 6 to \( \Lambda_{\text{cbn}}^\oplus \). Finitary soundness and finitary completeness both hold, exactly as in Theorem 17 and Theorem 20. The statement is the same, taking into account that now \( M \) is a closed term of \( \Lambda_{\text{cbn}}^\oplus \). As a consequence, we can

- on the one hand characterise AST and PAST in a uniform way, via tight typing, exactly as in Theorem 21 and Corollary 22;
- on the other hand characterise PAST via null typing, this time exactly like in Theorem 25.

As mentioned in Section 6.4.3, one can also obtain a (simpler) type system for AST by dropping from the typing all the information on the scaling factors.

8.2 Multidistributions vs. Distributions

In the design of any typing system, several choices are possible. Some are a matter of taste, some other are crucial. In this section, we discuss a choice we have implicitly made throughout the paper, namely the use of multidistributions in the typing system. One may legitimately wonder if we could use distributions of types instead of multidistributions. Actually, it turns out that multidistributions are necessary to obtain a perfect match between typing and termination. This choice is in fact crucial for completeness to hold in the call-by-value typing system. Let us see why.

Consider a term in the form \( \text{let } N = x \text{ in } M \). Since the argument \( N \) is typed with a multidistribution \( c = \langle p_k A_k \rangle_{k \in K} \), the continuation \( M \) must be able to receive any \( A_k \). Indeed, the typing rule \( \text{let} \) asks for type derivations having conclusion \( x : A_k \vdash M : b_k \) for each \( k \). Each value of \( k \) indeed corresponds to one of the possible probabilistic evolutions of \( N \), due to the use of multidistributions,
in which collapsing two elements of \( \mathcal{A}_k \) and \( \mathcal{A}_l \) in \( c \) is simply not possible. Going to distributions, thus allowing for such a collapse, would not be a problem for soundness, but we would lose the properties of weighted subject expansion (Lemma 19) on which completeness relies. We now see why by way of a concrete example.

**Example 29 (Weighted Subject Expansion relies on multidistributions).** Assume \( P \rightarrow \langle \frac{1}{2} P_1, \frac{1}{2} P_2 \rangle \). The claim of weighted subject expansion is that, given derivations \( \Pi_i \vdash^w P_i : b_i \) for each \( i \), we can obtain a derivation \( \Pi \vdash^w P : b \), where \( w \geq 1 + \sum \frac{1}{2} w_i \) and \( b = \frac{1}{2} b_1 \uplus \frac{1}{2} b_2 \). Weighted subject expansion is proved by induction on the structure of the reduction \( \rightarrow \). The key point is the \( \text{let} \, C \) rule. Let us focus on it. Consider \( P := (\text{let} \, N_1 \oplus N_2 = x \in M) \) and so \( P_1 := (\text{let} \, N_1 = x \in M) \), and consider the following type derivations for \( P_1 \) and \( P_2 \).

\[
\Pi_1 \vdash x : \mathcal{A} \vdash^w M : b_1 \vdash^w N_1 : \langle \mathcal{A} \rangle \\
\text{let} \quad \Pi_2 \vdash x : \mathcal{A} \vdash^w M : b_2 \vdash^w N_2 : \langle \frac{3}{4} \mathcal{A} \rangle \\
\text{let}
\]

By definition, \( P \rightarrow \langle \frac{1}{2} P_1, \frac{1}{2} P_2 \rangle \) is derived as follow:

\[
N \rightarrow \langle \frac{1}{2} N_1, \frac{1}{2} N_2 \rangle \\
(\text{let} \, N_1 \oplus N_2 = x \in M) \rightarrow \langle \frac{1}{2} (\text{let} \, N_1 = x \in M), \frac{1}{2} (\text{let} \, N_2 = x \in M) \rangle
\]

and we would like to derive a type derivation for \( P \) out of all this. By i.h., since \( N \rightarrow \langle \frac{1}{2} N_i \rangle_{i \in I} \), we can assume that there exists a derivation \( \Phi \vdash^w N : c \) such that \( v \geq 1 + \sum \frac{1}{2} v_i \) and \( c = \frac{1}{2} \langle \mathcal{A} \rangle \uplus \frac{1}{2} \langle \frac{3}{4} \mathcal{A} \rangle = \langle \frac{3}{4} \mathcal{A} \rangle \). And indeed, by collecting \( \Pi_1 \) and \( \Pi_2 \), we have a derivation which satisfies the claim

\[
(x : \mathcal{A} \vdash^w M : b_1)_{i \in \{1, 2\}} \vdash^w N : \langle \frac{3}{4} \mathcal{A} \rangle \\
\text{let}
\]

This is possible precisely because—due to the adoption of multidistributions—the two occurrences of \( \mathcal{A} \) are kept separated: notice that \( b_1 \) and \( b_2 \) may be very different types. If we worked with distributions, this information would be irremediably lost. By i.h., we would have just one derivation \( \Phi \vdash^w N : c \) where \( c = \frac{1}{2} \langle \mathcal{A} \rangle \uplus \frac{1}{2} \langle \frac{3}{4} \mathcal{A} \rangle = \langle \frac{3}{4} \mathcal{A} \rangle \). We would like to build the following derivation:

\[
\Pi \vdash^w N : \langle \frac{3}{4} \mathcal{A} \rangle \\
\text{let}
\]

How could we build \( \Pi \)? However? There is no way to merge the two derivations \( \Pi_1 \) and \( \Pi_2 \), so the type system would need to be substantially reengineered.

This issue only affects call-by-value evaluation, which is more complex than call-by-name, but also more expressive in a setting with effects. In CbN, choosing distributions would not impact the results, because evaluating a term before copying it (i.e. before using it in possibly many different ways) is simply impossible.

**9 RELATED WORK**

Systems of types for probabilistic programs exist in the literature. In particular, sized types [40], and linear dependent types [22] have been generalised to probabilistic programming languages, and have been proved to be sound methodologies for checking almost-sure termination [23] and positive almost-sure termination [4] in an higher-order setting. None of such systems is complete, however. Recently, Breuvart and Dal Lago [12] introduced systems of intersection types which are
sound and complete as a way of deriving the probability of convergence of terms in probabilistic
lambda-calculi. However, the number of reduction steps to normal form is not kept track of by
types, due to the nature of the intersection operator, which in Dal Lago and Breuvart’s system
is idempotent. Moreover, relying on distributions (instead of multidistributions) of types makes
call-by-value evaluation harder to deal with, and ultimately results in a rather convoluted set of
typing rules.

Intersection types have been pioneered by Coppo and Dezani [15, 16], and developed in a series
of papers in which various notions of termination for the $\lambda$-calculus were characterised, and the
relationship with denotational semantics was thoroughly investigated [6, 17, 19, 58]. They have also
been extended to calculi besides the $\lambda$-calculus, like $\lambda\mu$-calculi [66] or object calculi [28]. Besides
the already discussed work by Breuvart and Dal Lago [12], one should also mention the work by
d’e Liguoro and colleagues [29, 30] about filter models and intersection type assignment systems
for extensions of $\lambda$-calculi with nondeterministic choice operators, whose semantics is however
fundamentally different than that of the probabilistic choice operator we consider here: in the
former one observes may or must convergence (or combinations thereof), while here the notion of
observation is genuinely quantitative.

Non-idempotent intersection types have been known since the work by Gardner [34], studied in
connection with expansion variables by Carlier et al. [13], and further analysed in their relation to
normalisation by Mairson and Möller-Neergard [52]. The precise correspondence between non-
idempotent intersection type system derivations and the number of reduction steps necessary to
normalise the underlying term has been first noticed by De Carvalho [26], and further refined by
Bernadet and Lengrand [7], and later by Accattoli et al. [1], and [3], the latter being a source of
inspiration for this work in its reflecting weak notions of reduction inside intersection types. All
these contributions, however, deal with deterministic $\lambda$-calculi.

Formal verification techniques for probabilistic termination and complexity analysis are plentiful,
and ranges from model checking [32, 47] to abstract interpretation [50], to the ranking supermartin-
gales [14], to amortized analysis [53] to the interpretation method from term rewriting [5]. The only
methodology among these that, at least so far, has been employed for the analysis of higher-order
probabilistic programs is the one by Kobayashi et al. [47], which deals with probabilistic variations
on higher-order recursion schemes. Some of the ideas which we introduced in the paper are indeed
variations of similar ones from the imperative setting (e.g. the handling of expectations by way of
a quantity which decreases on the average). On the other hand, the presence of higher-order
functions forced us to develop new tools, since types must be more informative than just, say, a
ranking supermartingale: not only the value or the size of the input is crucial, but also the fact that
the input itself behaves well (given that it can potentially be used as a function).

The operational and denotational semantics of probabilistic $\lambda$-calculi have been studied thor-
oughly themselves, starting from the pioneering contributions by Sahed-Djaromi [61] and Jones
and Plotkin [41]. Noticeably, Ehrhard et al.’s probabilistic coherent spaces [31] can be presented as
a non-idempotent intersection type system which, being inherently semantic, is fundamentally
different from the one we have here: no result is given about the expected time to termination of
the interpreted terms, and results like those we proved in Section 7 would be much harder to get.

10 CONCLUSION

This paper introduces and studies non-idempotent intersection type assignment systems for proba-
bilistic $\lambda$-calculi, showing they can precisely characterise the expected runtime and the probability
of termination within a single framework, despite them having incomparable recursion-theoretic
difficulties, and thus an inherently different nature. The key ingredients are non-idempotency and
scaling. Noticeably, the same ideas work in the call-by-name and call-by-value paradigms.
The system of intersection types we have introduced in this work should be conceived as a tool for the theoretical analysis of a phenomenon, rather than as a proper verification technique: type inference is for obvious reasons highly undecidable. This does not mean, however, that the same necessarily holds in restricted calculi, as witnessed by the fruitful use of intersection types as a verification tool in subrecursive deterministic lambda-calculi [44, 48, 65]. As a consequence, it would be very interesting, e.g., to study which fragments of $\Lambda_{cbv}^\oplus$ and $\Lambda_{cbn}^\oplus$ are expressive enough to capture recursive markov chains [32], in which almost-sure termination is known to be decidable.

The absence of idempotency—an essential ingredient indeed—can be seen in two different forms, namely in intersection types, where union is not an idempotent operation, and in distribution types, which are taken as multidistributions and which thus do not form a barycentric algebra, precisely due to the failure of idempotency. A thorough study of this phenomenon, together with an analysis of the relationship between this work and the denotational semantics of probabilistic $\lambda$-calculi is outside the scope of this paper, but it is certainly something the authors would like to pursue in the foreseeable future.

REFERENCES

[1] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. Proc. of ICFP 2018, 2(ICFP):94:1–94:30, 2018.
[2] Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In 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] Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. Types by need. In Proc. of ESOP 2019, volume 11423 of LNCS, pages 410–439, 2019.
[4] Martin Avanzini, Ugo Dal Lago, and Alexis Ghyselen. Type-based complexity analysis of probabilistic functional programs. In Proc. of LICS 2019, pages 1–13, 2019.
[5] Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On probabilistic term rewriting. Sci. Comput. Program., 185, 2020.
[6] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. Journal of Symbolic Logic, 48(4):931–940, 1983.
[7] Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types and strong normalisation. Log. Methods Comput. Sci., 9(4), 2013.
[8] Patrick Billingsley. Probability and measure. John Wiley and Sons, New York, 1979.
[9] Ales Bizjak and Lars Birkedal. Step-indexed logical relations for probability. In Proc. of FoSSaCS, pages 279–294, 2015.
[10] Olivier Bournez and Florent Garnier. Proving positive almost sure termination under strategies. In Rewriting Techniques and Applications, RTA, pages 357–371, 2006.
[11] Pierre Brémaud. Discrete Probability Models and Methods. Springer, 2017.
[12] Flavien Breuvart and Ugo Dal Lago. On intersection types and probabilistic lambda calculi. In Proc. of PPDP 2018, pages 8:1–8:13, 2018.
[13] Sébastien Carlier, Jeff Polakow, J. B. Wells, and A. J. Kfoury. System E: expansion variables for flexible typing with linear and non-linear types and intersection types. In Proc. of ESOP 2004, volume 2986 of LNCS, pages 294–309. Springer, 2004.
[14] Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In Proc. of CAV 2013, volume 8044 of LNCS, pages 511–526. Springer, 2013.
[15] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for lambda-terms. Archiv für mathematische Logik und Grundlagenforschung, 19(1):139–156, 1978.
[16] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the $\lambda$-calculus. Notre Dame J. Formal Logic, 21(4):685–693, 10 1980.
[17] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Principal type schemes and lambda-calculus semantics. In To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pages 535–560. Academic Press, 1980.
[18] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. Math. Log. Q., 27(2-6):45–58, 1981.
[19] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Maddalena Zacchi. Type theories, normal forms and $d_{cbv}$-lambda-models. Inf. Comput., 72(2):85–116, 1987.
[20] Patrick Cousot. Types as abstract interpretations. In Proc. of POPL 1997, pages 316–331, 1997.
[21] Ugo Dal Lago and Beniamino Accattoli. Encoding Turing machines into the deterministic lambda-calculus. CoRR, abs/1711.10078, 2017. URL: http://arxiv.org/abs/1711.10078.
[22] Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. Log. Methods Comput. Sci., 8(4), 2011.
[23] Ugo Dal Lago and Charles Grellois. Probabilistic termination by monadic affine sized typing. ACM Trans. Program. Lang. Syst., 41(2):10:1–10:65, 2019.
[24] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In Proc. of POPL 2014, pages 297–308, 2014.
[25] Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. RAIRO - Theor. Inf. and Applic., 46(3):413–450, 2012.
[26] Daniel de Carvalho. Execution time of $\lambda$-terms via denotational semantics and intersection types. Math. Struct. Comput. Sci., 28(7):1169–1203, 2018. Available in preprint form from 2009 https://arxiv.org/abs/0905.4251.
[27] Karel De Leeuw, Edward F Moore, Claude E Shannon, and Norman Shapiro. Computability by probabilistic machines. Automata studies, 34:183–198, 1956.
[28] Ugo de’Liguoro. Characterizing convergent terms in object calculi via intersection types. In Proc. of TLCA 2001, volume 2044 of LNCS, pages 315–328. Springer, 2001.
[29] Ugo de’Liguoro and Adolfo Piperno. Non deterministic extensions of untyped lambda-calculus. Inf. Comput., 122(2):149–177, 1995.
[30] Mariangiola Dezani-Ciancaglini. Ugo de’Liguoro, and Adolfo Piperno. Filter models for a parallel and non deterministic lambda-calculus. In Proc. of MFCS 1993, pages 403–412, 1993.
[31] Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In Proc. of POPL 2014, pages 309–320. ACM, 2014.
[32] Kousha Etessami and Mihalis Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. J. ACM, 56(1):1:1–1:66, 2009.
[33] Luis Maria Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In Proc. of POPL 2015, pages 489–501, 2015.
[34] Philippa Gardner. Discovering needed reductions using type theory. In Proc. of TACS ’94, volume 789 of LNCS, pages 555–574. Springer, 1994.
[35] John Gill. Computational complexity of probabilistic Turing machines. SIAM Journal on Computing, 6(4):675–695, 1977.
[36] Jean-Yves Girard. Une extension de l’interpretation de gödel a l’analyse, et son application a l’elimination des coupures dans l’analyse et la theorie des types. In Proceedings of the Second Scandinavian Logic Symposium, volume 63 of Studies in Logic and the Foundations of Mathematics, pages 63 – 92. Elsevier, 1971.
[37] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. Journal of computer and system sciences, 28(2):270–299, 1984.
[38] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In UAI, pages 220–229, 2008.
[39] Jean-Yves Girard. Une extension de l’interpretation de gödel a l’analyse, et son application a l’elimination des coupures dans l’analyse et la theorie des types. In Proceedings of the Second Scandinavian Logic Symposium, volume 63 of Studies in Logic and the Foundations of Mathematics, pages 63 – 92. Elsevier, 1971.
[40] Achim Jung and Regina Tix. The troublesome probabilistic powerdomain of evaluations. In Proc. of LICS 1989, pages 186–195, 1989.
[41] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. Acta Informatica, 56(3):255–285, 2019.
[42] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In Andrew W. Appel and Alex Aiken, editors, POPL ’99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999, pages 161–174. ACM, 1999.
[43] A. J. Kfoury. A linearization of the lambda-calculus and consequences. J. Log. Comput., 10(3):411–436, 2000.
[44] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009, pages 416–428, 2009.
[45] Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. On the termination problem for probabilistic higher-order recursive programs. In Proc. of LICS 2019, pages 1–14, 2019.
Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal μ-calculus model checking of higher-order recursion schemes. In Proc. of LICS 2009, pages 179–188, 2009.

Dexter Kozen. Semantics of probabilistic programs. J. Comput. Syst. Sci., 22(3):328–350, 1981.

David Monniaux. An abstract analysis of the probabilistic termination of programs. In Proc. of SAS 2001, pages 111–126, 2001.

Rajeev Motwani and Prabhakar Raghavan. Randomized Algorithms. Cambridge University Press, 1995.

Peter Møller Neergaard and Harry G. Mairson. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In Proc. of ICFP 2004, pages 138–149, 2004.

Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: resource analysis for probabilistic programs. In Proc. of PLDI 2018, pages 496–512, 2018.

Piergiorgio Odifreddi. Classical Recursion Theory. Studies in Logic and Foundations of Mathematics, vol 125. Elsevier, 1989.

C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In Proc. of LICS 2006, pages 81–90, 2006.

Benjamin C. Pierce. Types and programming languages. MIT Press, 2002.

Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. Theor. Comput. Sci., 1(2):125–159, 1975.

Garrell Pottinger. A type assignment for the strongly normalizable lambda-terms. In To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pages 561–577. Academic Press, 1980.

Martin L. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

Michael O Rabin. Probabilistic automata. Information and control, 6(3):230–245, 1963.

N. Saheb-Djahromi. Probabilistic LCF. In Proc. of MFCS 1978, volume 64 of LNCS, pages 442–451, 1978.

Eugene S. Santos. Probabilistic Turing machines and computability. Proceedings of the American Mathematical Society, 22(3):704–710, 1969.

Morten Heine Sørensen and Pawel Urzyczyn. Lectures on the Curry-Howard Isomorphism. Studies in Logic and Foundations of Mathematics, vol 149. Elsevier, 1989.

David Tolpin, Jan-Willem van de Meent, and Frank D. Wood. Probabilistic programming in Anglican. In Proc. of ECML PKDD 2015, volume 9286 of LNCS, pages 308–311. Springer, 2015.

Takeshi Tsukada and Naoki Kobayashi. An intersection type system for deterministic pushdown automata. In Proc. of TCS 2012, pages 357–371, 2012.

Steffen van Bakel, Franco Barbanera, and Ugo de’Liguoro. Characterisation of strongly normalising lambda-mu-terms. In Proc. of ITRS 2012, volume 121 of EPTCS, pages 1–17, 2012.

Christopher Wadsworth. Some unusual 𝜆-calculus numeral systems. In J.P. Seldin and J.R. Hindley, editors, To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press, 1980.
A PROOFS OF SUBJECT REDUCTION AND SUBJECT EXPANSION

A.1 Proof of Weighted Subject Reduction
As usual, the proof of subject reduction relies on a substitution lemma.

**Lemma 30 (Substitution Lemma).** If exist derivations \( \Pi \triangleright \Gamma, z : C \vdash^w M : T, \) and \( \Phi \triangleright^\Phi V : C, \) then exists a derivation \( \Pi' \triangleright \Gamma \vdash^w M\{V/z\} : T. \) Moreover

1. \( w' = w + v \) and
2. \( |\Pi'| \leq |\Pi| + |\Phi|. \)

**Proof.** The proof is by induction on the derivation \( \Pi; \) we examine the last rule. We write only the key cases.

- Rule \( \text{let}. \) Assume \( \Pi \) is as follows

\[
\frac{z : C_k, \Delta_k, x : \mathcal{A}_k \vdash^w b_k}_{z : C_0 \vdash_k p_k.C_k, \Gamma \vdash_k \Delta_k \vdash^w \vdash N \vdash_k p_k.b_k}
\]

\[
\text{let } N = x \text{ in } M : \vdash_k p_k.b_k
\]

and \( \Phi \triangleright^\Phi V : C = C_0 \vdash_k p_k.C_k. \) By partition and Scaling (Lemma 11), we have \( v = v_0 + \sum_k v_k \) and

\[
\vdash_k V : C_0 \left\langle \frac{v_k}{p_k} \vdash_k V : C_k \right\rangle
\]

By i.h., we obtain the following derivation

\[
\frac{\Delta_k, x : \mathcal{A}_k \vdash^w b_k}{\Gamma \vdash_k \Delta_k \vdash^w \vdash \vdash N \vdash_k p_k.b_k}
\]

where \( w' = (w_0 + v_0) + \sum_k p_k(w_k + \frac{v_k}{p_k}) + 1. \)

We verify that claims 1. and 2. hold.

1. \( w' = w_0 + \sum_k p_k(w_k) + 1 + v_0 + \sum_k p_k\left(\frac{v_k}{p_k}\right) = w + v \)
2. (immediate).

- Rules \( ! \) and \( \oplus \) also use partition and Scaling (Lemma 11).

- Rules \( \lambda \) and \( @ \) are as usual.

\[\square\]

**Lemma** (16. Weighted Subject Reduction). Suppose that \( \Pi \triangleright P : b, \) with \( w > 0, \) and that \( P \rightarrow (q_i P_i)_{i \in I}. \) Then for every \( i \in I \) there exists a derivation \( \Pi_i \) such that \( \Pi_i \triangleright P_i : b_i, \) and \( |\Pi| > |\Pi_i|. \) Moreover:

1. \( b = \big|_i q_i b_i \)
2. \( w = 1 + \sum_i q_i w_i \)

**Proof.** Observe that \( w \neq 0 \) implies that the last rule of the derivation \( \Pi \) is not a ZERO-rule. The proof is by induction on the definition of the reduction step \( P \rightarrow (q_i P_i)_{i \in I}. \)

- \( \beta. \) Assume \( P := (\lambda x.M)V \) and

\[
(\lambda x.M)V \rightarrow (M\{V/x\}) \beta
\]

We examine the derivation \( \Pi, \) and conclude by Substitution Lemma.

, Vol. 1, No. 1, Article . Publication date: October 2020.
• LetV. Assume $P := (\text{let } V = x \text{ in } M) \text{ and } P \rightarrow \langle 1P_0 \rangle$ as follows

\[
\text{let } V = x \text{ in } M \rightarrow \langle M\{V/x\} \rangle \quad \text{letV}
\]

By assumption, there exists $\Pi \triangleright \vdash P : a$, which must have the following shape

\[
x : C \vdash M : a \quad \vdash \quad V : \langle C \rangle \quad \vdash \quad \text{let}
\]

By Substitution Lemma, there exists $\Pi' \triangleright \vdash M\{V/x\} : a$. Observe that $I = \{0\}$ is a singleton, and $q_0 = 1$. Conditions 1., 2., 3. are all satisfied.

• $\oplus$. Assume $P := P_1 \oplus P_2$ and $P \rightarrow \langle \frac{1}{2}P_1, \frac{1}{2}P_2 \rangle$ as follows

\[
P_1 \oplus P_2 \rightarrow \langle \frac{1}{2}P_1, \frac{1}{2}P_2 \rangle \oplus
\]

By assumption, there exists a type derivation $\Pi \triangleright \vdash P_1 \oplus P_2 : a$, which must have the following shape

\[
\vdash P_1 : a_1 \quad \vdash P_2 : a_2 \quad \vdash \quad P_1 \oplus P_2 : a = (\frac{1}{2}a_1 | \frac{1}{2}a_2)
\]

All the points in the claim hold.

• LetC. Assume $P := (\text{let } N = x \text{ in } M) \text{ and } P \rightarrow \langle q_i P_i \rangle$, with $P_i := (\text{let } N_i = x \text{ in } M)$, according to

\[
(\text{let } N = x \text{ in } M) \rightarrow \langle q_i(\text{let } N_i = x \text{ in } M) \rangle \quad \text{letC}.
\]

By assumption, there exists a type derivation $\Pi \triangleright \vdash P : b$, which must have the following shape

\[
(x : A_k \vdash M : b_k)_{k \in K} \quad \Phi \triangleright \vdash N : a = \langle p_k A_k \rangle_{k \in K} \quad \text{let}
\]

By $i.h.$, there exist type derivations $\Phi_i \triangleright \vdash N_i : a_i$ such that $|\Phi| > |\Phi_i|$, $\forall i \in I$ and moreover

1. $a = \biguplus_{i \in I} q_i a_i$;
2. $\nu = 1 + \sum_{i \in I} q_i \nu_i$;

By point 1. above, $\biguplus_{i \in I} q_i a_i = \langle p_k A_k \rangle_{k \in K}$. As $K$ is the index set of $a$, let $K_i$ be the index set of $a_i (i \in I)$ (notice that for some $i$, it may be possible $K_i = \emptyset$). We have $K = \bigcup_{i \in I} K_i$ and $q_i a_i = \langle p_k A_k \rangle_{k \in K_i}$; therefore $a_i = \left(\frac{p_k A_k}{q_i} \right)_{k \in K_i}$. For each $i \in I$ we obtain the following type derivation $\Pi_i$:

\[
(x : A_k \vdash M : b_k)_{k \in K_i} \quad \Phi_i \triangleright \vdash N_i : a_i = \left(\frac{p_k A_k}{q_i} \right)_{k \in K_i} \quad \text{let}
\]

\[
\biguplus_{i \in I} q_i b_i = \biguplus_{i \in I} q_i (\biguplus_{k \in K_i} \frac{p_k A_k}{q_i} b_k) = \biguplus_{i \in I} (\biguplus_{k \in K_i} q_i \frac{p_k A_k}{q_i} b_k) = \biguplus_{k \in K} p_k b_k = b
\]

We check that point 1. and 2. of the claim are verified.

(1) $\biguplus_{i \in I} q_i b_i = \biguplus_{i \in I} q_i (\biguplus_{k \in K_i} \frac{p_k A_k}{q_i} b_k) = \biguplus_{i \in I} (\biguplus_{k \in K_i} q_i \frac{p_k A_k}{q_i} b_k) = \biguplus_{k \in K} p_k b_k = b$
Intersection Types and (Positive) Almost-Sure Termination

\(1 + \sum_{i \in I} q_i w_i\)

\[(2)\]

\[= 1 + \sum_{i \in I} q_i \sum_{k \in K} p_k q_i + \sum_{i \in I} q_i\]

\[= (1 + \sum_{i \in I} q_i q_i) + \sum_{k \in K} q_k \sum_{i \in I} p_k q_i + \sum_{i \in I} q_i\]

\[= v + \sum_{k \in K} p_k q_k + \sum_{i \in I} q_i\]

\[= v + \sum_{k \in K} p_k q_k + 1\]

where \(\sum_{i \in I} q_i = 1\) is by definition of \(P \rightarrow \langle q_i P_i \rangle_{i \in I}\).

\(\square\)

### A.2 Proof of Weighted Subject Expansion

The proof of Subject expansion relies on an anti-substitution lemma.

**Lemma 31 (Anti-substitution).** Assume \(\Phi \triangleright \Gamma \vdash M\{V/x\} : a\), where \(V\) is closed. Then there exist:

1. an intersection type \(C\);
2. a derivation \(\Phi_1 \triangleright \Gamma, x : C \vdash w_1\)
3. a derivation \(\Phi_2 \triangleright w_2 \vdash V : C\)

such that

- \(a = \biguplus_i q_i a_i\)
- \(w = w_1 + w_2\)

**Proof.** By induction on \(\Pi\). The proof is routine. \(\square\)

**Property 32.** If \(\Pi \triangleright w+1 \vdash \text{let } N = x \text{ in } M : 0\), then \(\Pi\) has either of the following shapes:

\[\Pi_i \triangleright w+i \vdash \text{let } N = x \text{ in } M : 0\]

\[\Pi_i \triangleright w+i+\sum_{k \in K} p_k q_k + 1 \vdash \text{let } N_l = x \text{ in } M : 0\]

**Lemma** (19. Weighted Subject Expansion). Let \(P\) be a closed term. Assume that \(P \rightarrow \langle q_i P_i \rangle_{i \in I}\) and that for each \(i \in I\), \(\Pi_i \triangleright w_i \vdash P_i : a_i\). Then, there exists a single derivation \(\Pi \triangleright w \vdash P : a\) such that \(a = \biguplus_i (q_i a_i)\) and \(w \geq 1 + \sum q_i w_i\).

**Proof.** The proof is by induction on the definition of the reduction step \(P \rightarrow \langle q_i P_i \rangle_{i \in I}\).

- \(\beta\). Let \(P := (\lambda x.M)V\) and

\[\beta\]

By assumption, there exists \(\Pi' \triangleright w \vdash M\{V/x\} : a\). By anti-substitution Lemma, there exist derivations \(\Phi_1, \Phi_2\) which allow for the following inference, where \(w' = w_1 + w_2\)
\[
\begin{align*}
\Phi_1 \triangleright x : C &\vdash M : a \\
\frac{1 + \alpha}{\vdash \lambda x . M : C \rightarrow a} &\\
\frac{1 + \alpha}{\vdash \lambda x . M : \{C \rightarrow a\}} &\\
\frac{1 + \alpha + \alpha}{\vdash V : C} &
\end{align*}
\]

\[\Phi_2 \triangleright \vdash V : C\]

\[\frac{1 + \alpha + \omega}{\vdash (\lambda x . M)V : a}\]

- \texttt{letV}. Let \(P := (\texttt{let } V = x \texttt{ in } M)\) and

\[\texttt{let } V = x \texttt{ in } M \rightarrow \langle M\{V/x\} \rangle\]

By assumption, there exists \(\Pi' \triangleright \vdash M\{V/x\} : a\). By anti-substitution Lemma, there exist derivations \(\Phi_1, \Phi_2\) which allow for the following inference, where \(\omega' = \omega_1 + \omega_2\)

\[\Phi_1 \triangleright \frac{\omega_1}{\vdash V : C} \quad \Phi_2 \triangleright \frac{\omega_2}{\vdash V : \langle C \rangle} \quad \vdash \texttt{let } V = x \texttt{ in } M : a\]

- \texttt{⊕}. Let \(P := P_1 \oplus P_2\) and

\[\frac{\vdash P_1 : a_1 \quad \vdash P_2 : a_2}{\vdash P_1 \oplus P_2 : \frac{1}{2}a_1 | \frac{1}{2}a_2} \quad \oplus\]

By assumption, for each \(i \in I\), there exists \(\Pi_i \triangleright \vdash P_i : a_i\). Therefore, we obtain

\[\frac{\vdash P_1 : a_1 \quad \vdash P_2 : a_2}{\vdash P_1 \oplus P_2 : \frac{1}{2}a_1 | \frac{1}{2}a_2} \quad \oplus\]

- \texttt{letC}. Let \(P := (\texttt{let } N = x \texttt{ in } M), P_i := (\texttt{let } N_i = x \texttt{ in } M)\), and

\[\frac{\vdash \langle q_i N_i \rangle_{i \in I}}{(\texttt{let } N = x \texttt{ in } M) \rightarrow \langle q_i (\texttt{let } N_i = x \texttt{ in } M) \rangle} \quad \texttt{letC}\]

By assumption, for each \(i \in I\), there exists a derivation \(\Pi_i \triangleright \vdash P_i : a_i\). Each \(\Pi_i\) has one of the following shapes:

1. \[
\frac{}{\vdash (\texttt{let } N_i = x \texttt{ in } M) \vdash 0} \quad \text{ZERO}
\]

2. \[
\frac{\vdash x : \mathcal{A}_k \vdash M : b_{k_i} \mid k \in K_i \quad \vdash N_i : \langle \mathcal{A}_k \rangle_{k \in K_i} := c_i}{\Pi_i \triangleright \vdash (\texttt{let } N_i = x \texttt{ in } M) \vdash \langle \mathcal{A}_k \rangle_{k \in K_i} \mid p_k b_k \vdash 0} \quad \texttt{let}
\]

Note that \(K_j\) may be \(0\) for some \(i\), and that \(b_k\) may be \(0\) for some \(k\).

Let us examine the three sub-cases.

(i) Suppose that all \(\Pi_i\) are as (1). The following \(\Pi\) satisfies the claim

\[\frac{}{\vdash N : 0} \quad \frac{\vdash \text{let } N = x \texttt{ in } M : 0}{\vdash 1}\]


(iii) Suppose all \( \Pi_i \) are as (2). By i.h., since \( N \rightarrow (q_i N_i)_{i \in I} \), there exists a derivation \( \Phi \vdash^a N : c \) such that \( v \geq 1 + \sum q_i v_i \) and \( c = \|_{i \in I} (q_i \langle p_k \mathcal{A}_k \rangle_{k \in K_i}) \), that is \( c = \|_{i \in I} (q_i p_k \mathcal{A}_k)_{k \in K_i} \).

Therefore, by collecting all premisses \( \Theta_k \) for \( k \in \bigcup_{i \in I} K_i \), we obtain

\[
(x : \mathcal{A}_k \vdash^a M : b_k)_{k \in \bigcup_{i \in I} K_i} \quad \Phi \vdash^a N : \|_{i \in I} (q_i p_k \mathcal{A}_k)_{k \in K_i}
\]

let

\[
\Pi \vdash^a \text{let } N = x \text{ in } M : \|_{i \in I} q_i (\|_{k \in K_i} p_k b_k)
\]

where \( w = v + \sum_i (\sum_{k \in K_i} q_i p_k v_k) + 1 \). Since \( v \geq 1 + \sum q_i v_i \) the claim holds because:

\[
1 + \sum q_i w_i = \\
1 + \sum q_i v_i + \sum q_i (\sum_{k \in K_i} p_k v_k) + \sum q_i = \\
(1 + \sum q_i v_i) + \sum q_i (\sum_{k \in K_i} p_k v_k) + 1 \\
\leq v + \sum q_i (\sum_{k \in K_i} p_k v_k) + 1 \\
= w
\]

where \( \sum_{i \in I} q_i = 1 \) because \( P \rightarrow (q_i P_i)_{i \in I} \).

(iii) Suppose that some, but not all, \( \Pi_i \) are as (1). We replace those \( \Pi_i \) with the following other type derivation of \( N_i \)

\[
\begin{align*}
\vdash^a & N_i : 0 \\
\vdash^a & \text{let } N_i = x \text{ in } M : 0 \\
\end{align*}
\]

and use the argument (ii). In doing so, we are overapproximating the weight \( w_i' = 0 \) of the original derivation \( \Pi_i \) with \( w_i' = 1 \), which is fine, since ultimately we want to obtain a derivation \( \Pi \) whose weight is \( w \geq 1 + \sum_i q_i w_i \).

Case (iii) is exactly the reason why we have \( w \geq 1 + \sum q_i w_i \) instead of \( w = 1 + \sum q_i w_i \).

\(\square\)

\section*{B \ MORE VARIATIONS AND DESIGN CHOICES}

\subsection*{B.1 \ How much to increment the counter?}

As we already observed, in the typing system, the \texttt{let}-rule increase the counter of 1 (to reflect a \texttt{letC}-step) even when such a step never happen. As a consequence, in Weighted Subject Reduction, and therefore in the Completeness Theorem, there is a slight asymmetry: for \( k \in \text{Nat} \), there is a proof \( \Pi \vdash^w M : a \) such that (1) \( \| a \| = \text{PTerm}_k(M) \) and (2) \( w \geq \text{ETime}_k(M) \).

It is indeed possible to give a rule which accurately reflects all steps, as follows.

\[
\begin{align*}
(\Delta_k, x : \mathcal{A}_k \vdash^w M : b_k)_{k \in K} & \quad \Gamma \vdash^w N : \langle p_k \mathcal{A}_k \rangle_{k \in K} \\
& \quad \Gamma \vdash^w \text{let } N = x \text{ in } M : \|_{k \in K} p_k b_k
\end{align*}
\]

We have decided to privilege simplicity.

Choosing the accurate counter \( w + \sum_k p_k w_k + \sum_k p_k \) would lend, both in the Completeness Theorem and in Weighted Subject Expansion, an equality for both \( a \) and \( w \).

\begin{example}[Two Ways of Counting]
This example pinpoints the technical difference between the two design choices for the counter in the \texttt{let}-rule, and the consequence in Weighted Subject Reduction. The difference appears when we study the \texttt{letC} case in the proof of Subject Reduction, precisely the case (iii).
\end{example}
Consider the case where \( \text{let } N = x \text{ in } M \rightarrow (\frac{1}{2} \text{ let } N_1 = x \text{ in } M, \frac{1}{2} \text{ let } N_2 = x \text{ in } M) \). \( \Pi_1 \) and \( \Pi_2 \) are

\[
\Pi_1 \vdash^{v_1=0} \text{let } N_1 = x \text{ in } M : 0 \\
\Pi_2 \vdash^{v_2=1} \text{let } N_2 = x \text{ in } M : 0
\]

Therefore \( \frac{1}{2}v_1 + \frac{1}{2}v_2 + 1 = \frac{u}{2} + \frac{1}{2} + 1 \).

Desired claim: there exists a derivation \( \Pi \vdash w \text{ let } N = x \text{ in } M : 0 \) where \( w = \frac{1}{2}v_1 + \frac{1}{2}v_2 + 1 \)

1. **Counter +1.** \( \Pi_1 \) and \( \Pi_2 \) are as follows:

\[
\begin{array}{c}
0 \vdash \text{let } N_1 = x \text{ in } M : 0 \\
\frac{u+1}{2} \vdash \text{let } N_1 = x \text{ in } M : 0
\end{array}
\]

By i.h. (using \( 0 \vdash N_1 : 0 \)) there is a derivation of \( \frac{u+1}{2} \vdash N : 0 \), from which we obtain

\[
\frac{w-u+1}{2} \vdash \text{let } N = x \text{ in } M : 0
\]

That is, \( w > \frac{1}{2}v_1 + \frac{1}{2}v_2 + 1 \)

2. **Accurate counter.** \( \Pi_1 \) and \( \Pi_2 \) are as follows:

\[
\begin{array}{cc}
0 \vdash N_1 : 0 \\
\frac{u+1}{2} \vdash N_2 : 0
\end{array}
\]

\[
\begin{array}{c}
0 \vdash \text{let } N_1 = x \text{ in } M : 0 \\
\frac{u+1}{2} \vdash \text{let } N_2 = x \text{ in } M : 0
\end{array}
\]

By i.h. (using \( 0 \vdash N_1 : 0 \)) there is a derivation of \( \frac{u+1+1}{2} \vdash N : 0 \), from which we obtain

\[
\frac{w-u+1+1}{2} \vdash \text{let } N = x \text{ in } M : 0
\]

That is, \( w = \frac{1}{2}v_1 + \frac{1}{2}v_2 + 1 \).