Intersection Types and (Positive) Almost-Sure Termination

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

Randomised higher-order computation can be seen as being captured by a λ-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 λ-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.

CCS Concepts: • **Theory of computation** → **Type theory**; **Lambda calculus**; **Probabilistic computation**.

Additional Key Words and Phrases: almost-sure termination, expected time, type systems, intersection types

ACM Reference Format:
Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. 2021. Intersection Types and (Positive) Almost-Sure Termination. *Proc. ACM Program. Lang.* 5, POPL, Article 32 (January 2021), 32 pages. https://doi.org/10.1145/3434313

1 INTRODUCTION

The study and analysis of randomised computation is almost as old as theoretical computer science itself [De Leeuw et al. 1956; Rabin 1963; Santos 1969]. 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 [Motwani and Raghavan 1995], but also secure cryptographic primitives (e.g. public-key cryptosystems [Goldwasser and Micali 1984]), 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 [Kozen 1981; Saheb-Djahromi 1978], but has shown a strong impetus in the

*Also with INRIA Sophia Antipolis.

Authors' addresses: Ugo Dal Lago, Università di Bologna, Italy, ugo.dallago@unibo.it; Claudia Faggian, Université de Paris, IRIF, CNRS, Paris, France, France, claudia.faggian@irif.fr; Simona Ronchi Della Rocca, Università di Torino, Italy, ronchi@di.unito.it.

2475-1421/2021/1-ART32
https://doi.org/10.1145/3434313

last ten years, due to progress in so-called bayesian programming languages [Goodman et al. 2008; Tolpin et al. 2015], 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 [Bizjak and Birkedal 2015], or by way of coinduction [Dal Lago et al. 2014] 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 [Jones and Plotkin 1989; Jung and Tix 1998], and has been solved in a completely satisfactory way only relatively recently [Ehrhard et al. 2014; Goubault-Larrecq 2015].

Types and Verification. Verification of deterministic higher-order programs can be carried out in many ways, including model checking [Ong 2006], abstract interpretation [Cousot 1997], and type systems [Pierce 2002]. 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 [Hughes et al. 1996; Sørensen and Urzyczyn 1989]. This idea has been brought to its extreme consequences by the line of work on intersection types [Coppo and Dezani-Ciancaglini 1978; Coppo et al. 1981], 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 [Barendregt et al. 1983], and also to support program verification in subrecursive languages [Kobayashi 2009].

On Probabilistic Termination's 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*[1] 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 [Kaminski et al. 2019]: both almost-sure termination and positive almost-sure termination are not recursively enumerable, and have *incomparable* recursion-theoretic statuses, the former being Π^0_2-complete, the latter being Σ^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 [Kobayashi et al. 2019], 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 [Ong 2006]. 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 λ-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?

[1] The term was introduced in [Bournez and Garnier 2006], but the requirement that the program be expected to terminate is natural and fundamental, and was already present in [Saheb-Djahromi 1978].

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 λ-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 $\varepsilon > 0$ there is a type derivation for M which gives an ε-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 [Dal Lago and Grellois 2019], intersection types [Coppo and Dezani-Ciancaglini 1978], and non-idempotency [de Carvalho 2018]. 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 choice. 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 enumerable) system of types for a probabilistic λ-calculus is either unsound or incomplete as a way to *precisely* verify termination properties of pure λ-terms. In other words, one cannot do better than what we do. Remarkably, all results we give in this paper hold for both call-by-value and call-by-name evaluation, but we prefer to give all the details of the a system of the former kind, arguably a more natural one in presence of effects.

An extended version of this paper with proofs and more details is available [Dal Lago et al. 2020].

2 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) λ-calculus $\Lambda^{\mathtt{cbv}}$ in A-normal form. Values and terms are generated through the following grammars:

$$V ::= x \mid \lambda x.M \qquad \textbf{Values}, \mathcal{V}^{\mathtt{cbv}}$$

$$M ::= V \mid VV \mid \mathtt{let}\ x = M\ \mathtt{in}\ M \qquad \textbf{Terms}, \Lambda^{\mathtt{cbv}}$$

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

This language can be seen as a fragment of Plotkin's CbV λ-calculus [Plotkin 1975] in which the latter can be faithfully embedded[2]. As such, the calculus is easily seen to be Turing-universal, and termination is thus an undecidable—although recursively enumerable—problem. How could we

[2] an application MN becomes the term $\mathtt{let}\ x = M\ \mathtt{in}\ \mathtt{let}\ y = N\ \mathtt{in}\ xy$.

$$\mathtt{A} ::= \alpha \mid \mathtt{A} \to \mathtt{A}$$

$$\frac{}{\Gamma, x : \mathtt{A} \vdash x : \mathtt{A}} \qquad \frac{\Gamma, x : \mathtt{A} \vdash M : \mathtt{B}}{\Gamma \vdash \lambda x.M : \mathtt{A} \to \mathtt{B}}$$

$$\frac{\Gamma \vdash V : \mathtt{A} \to \mathtt{B} \quad \Gamma \vdash W : \mathtt{A}}{\Gamma \vdash VW : \mathtt{B}} \qquad \frac{\Gamma \vdash N : \mathtt{A} \quad \Gamma, x : \mathtt{A} \vdash M : \mathtt{B}}{\Gamma \vdash \mathtt{let}\ x = N\ \mathtt{in}\ M : \mathtt{B}}$$

Fig. 1. Simple Types

$$\mathtt{A} ::= \mathcal{A} \to \mathcal{A}$$
$$\mathcal{A} ::= \{\mathtt{A}_1, \ldots, \mathtt{A}_n\}$$

$$\frac{}{\Gamma, x : \mathcal{A} \vdash x : \mathcal{A}} \qquad \frac{\Gamma, x : \mathcal{A} \vdash M : \mathcal{B}}{\Gamma \vdash \lambda x.M : \mathcal{A} \to \mathcal{B}} \qquad \frac{\{\Gamma_i \vdash V : \mathtt{A}_i\}_{i \in I}}{\bigcup_i \Gamma_i \vdash V : \{\mathtt{A}_i\}_{i \in I}}$$

$$\frac{\Gamma \vdash V : \mathcal{A} \to \mathcal{B} \quad \Gamma \vdash W : \mathcal{A}}{\Gamma \vdash VW : \mathcal{B}} \qquad \frac{\Gamma \vdash N : \mathcal{A} \quad \Gamma, x : \mathcal{A} \vdash M : \mathcal{B}}{\Gamma \vdash \mathtt{let}\ x = N\ \mathtt{in}\ M : \mathcal{B}}$$

Fig. 2. Idempotent Intersection Types for $\Lambda^{\mathtt{cbv}}$

compositionally guarantee termination of those λ-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 $\Lambda^{\mathtt{cbv}}$ is in Figure 1, where types are either an atom α or an arrow type $\mathtt{A} \to \mathtt{B}$. A simple reducibility-like argument indeed shows 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 $\mathtt{A}$ and the type $\mathtt{A} \to \mathtt{B}$.

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 $\mathbb{F}$ [Girard 1971] dramatically increases the expressive power of simple types by way of a form of (second-order) quantification: the type $\forall\alpha.\mathtt{A}$ stands for all types which can be obtained as formal instances of $\mathtt{A}$. Parametric 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 $\mathcal{A} = \{\mathtt{A}_1, \ldots, \mathtt{A}_n\}$, and take arrow types as expressions in the form $\mathcal{A} \to \mathcal{B}$. The type $\mathcal{A}$ can be assigned to terms which have type $\mathtt{A}_j$ *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 *multi*sets, i.e. $\mathcal{A} = [\mathtt{A}_1, \ldots, \mathtt{A}_n]$. 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 $\overset{w}{\vdash} M : []$ (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 [Accattoli et al. 2019].

$$\mathtt{A} ::= \mathcal{A} \to \mathcal{A} \qquad \mathcal{A} ::= [\mathtt{A}_1, \ldots, \mathtt{A}_n]$$

$$\frac{}{x : \mathcal{A} \overset{0}{\vdash} x : \mathcal{A}} \qquad \frac{\Gamma, x : \mathcal{A} \overset{w}{\vdash} M : \mathcal{B}}{\Gamma \overset{w+1}{\vdash} \lambda x.M : \mathcal{A} \to \mathcal{B}} \qquad \frac{\{\Gamma_i \overset{w_i}{\vdash} V : \mathtt{A}_i\}_{i \in I}}{\uplus_i \Gamma_i \overset{\sum_i w_i}{\vdash} V : [\mathtt{A}_i]_{i \in I}}$$

$$\frac{\Gamma \overset{w}{\vdash} V : \mathcal{A} \to \mathcal{B} \quad \Delta \overset{v}{\vdash} W : \mathcal{A}}{\Gamma \uplus \Delta \overset{w+v}{\vdash} VW : \mathcal{B}} \qquad \frac{\Gamma \overset{w}{\vdash} N : \mathcal{A} \quad \Delta, x : \mathcal{A} \overset{v}{\vdash} M : \mathcal{B}}{\Gamma \uplus \Delta \overset{w+v+1}{\vdash} \mathtt{let}\ x = N\ \mathtt{in}\ M : \mathcal{B}}$$

Fig. 3. Non-Idempotent Intersection Types for $\Lambda^{\mathtt{cbv}}$

2.2 Typing Termination in a Probabilistic Setting

How about probabilistic λ-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 to I with 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, and that almost-sure termination does *not* imply *positive* almost-sure termination.

[3] Accommodating 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.

Example 2. *For every natural number n, let $\overline{n}$ be an encoding of it as a λ-term, and let $SUCC$ and EXP be terms which encode the successor and the exponential function, respectively.*

- *Consider the term CC where $C = \lambda x.(\mathbf{succ}(xx) \oplus \overline{0})$, and where $\mathbf{succ}(M)$ is syntactic sugar for $(\mathtt{let}\ z = M\ \mathtt{in}\ SUCC\ z)$. Note that $\mathbf{succ}(\overline{n})$ reduces to $\overline{n+1}$ in constant time s ($s \in \mathbb{N}$), of course depending on the chosen encoding. The program CC reduces—at the limit—to each natural number $\overline{n}$ with probability $\frac{1}{2^{n+1}}$. It is clear that CC is* AST*, and it is easy to check that it is also* PAST*; indeed it is expected to terminate in $4 + s$ steps. However its reduction graph, contrarily to the one of DD, involves* infinitely many normal forms.
- *The term $\mathbf{exp}(CC)$, where $\mathbf{exp}(M)$ is syntactic sugar for $(\mathtt{let}\ z = M\ \mathtt{in}\ EXP\ z)$, 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 λ-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_1D_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_2D_2, each with equal probability. These two observations inform how non-idempotent intersection types can be generalised to a λ-calculus with probabilistic choice. Indeed, the multisets $\mathcal{A}$ and $\mathcal{B}$ in an arrow type $\mathcal{A} \to \mathcal{B}$ 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: $1 + \sum_{i=1}^{\infty} \frac{1}{2}$. 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 [Kaminski et al. 2019], 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 λ-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 $\Lambda_{\oplus}^{\mathtt{cbv}}$. 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 $\mathbb{X}$) as $[\mathtt{a}_j]_{j\in J}$, 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 $\uplus$.

Distributions. Let Ω be a *countable* set. A function $\mu : \Omega \to [0, 1]$ 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 $\mathcal{D}(\Omega)$ for the set of subdistributions on Ω, equipped with the standard pointwise partial order relation : $\mu \le \rho$ if $\mu(\omega) \le \rho(\omega)$ for each $\omega \in \Omega$. The *support* of μ is the set $\{\omega \mid \mu(\omega) > 0\}$.

Multidistributions. Suppose $\mathbb{X}$ is a countable set and let $\mathtt{m}$ be a finite multiset of pairs of the form pM, with $p \in (0, 1]$, and $M \in \mathbb{X}$. Then $\mathtt{m} = [p_i M_i]_{i\in I}$ is said to be a *multidistribution on* $\mathbb{X}$ if $\|\mathtt{m}\| := \sum_{i\in I} p_i \le 1$. For multidistributions, we use the notation $\mathtt{m} = \langle p_i M_i\rangle_{i\in I}$. The empty multidistribution is indicated as $\mathbf{0}$ (note that $\|\mathbf{0}\| = 0$). We denote by $\mathcal{M}(\mathbb{X})$ the set of all multidistributions on $\mathbb{X}$. We indicate the multidistribution $\langle 1M\rangle$ simply as $\langle M\rangle$. The (disjoint) sum of multidistributions is denoted as $\boxplus$, and is a *partial* operation. The product $q\cdot\mathtt{m}$ of a scalar $q < 1$ and a multidistribution $\mathtt{m}$ is defined pointwise: $q \cdot \langle p_1 M_1, \ldots, p_n M_n\rangle = \langle (qp_1)M_1, \ldots, (qp_n)M_n\rangle$. Intuitively, a multidistribution $\mathtt{m}$ is an intensional representation of a probability distribution: multidistributions do not satisfy the equation $\mathtt{m} = p \cdot \mathtt{m} \boxplus (1 - p) \cdot \mathtt{m}$. 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 $\Lambda_{\oplus}^{\mathtt{cbv}}$

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

$$V ::= x \mid \lambda x.M \qquad \textbf{Values}, \mathcal{V}_{\oplus}^{\mathtt{cbv}}$$

$$M ::= V \mid VV \mid M \oplus M \mid \mathtt{let}\ x = M\ \mathtt{in}\ M \qquad \textbf{Terms}, \Lambda_{\oplus}^{\mathtt{cbv}}$$

where x ranges over a countable set of *variables*. $\Lambda_{\oplus}^{\mathtt{cbv}}$ and $\mathcal{V}_{\oplus}^{\mathtt{cbv}}$ denote respectively the set of terms and of values. Free and bound variables are defined as usual, while $M\{N/x\}$ denotes the term obtained from the capture-avoiding substitution of N for all the free occurrences of x in M. As usual, a *program* is a closed term. Throughout the paper we frequently use the following terms as examples:

$$I := \lambda x.x; \qquad \Delta := \lambda x.xx; \qquad D := \lambda x.(xx \oplus I).$$

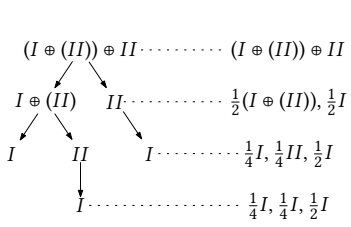
The program $\Delta\Delta$ is the paradigmatic diverging term, while DD is our running example.

3.3 The Operational Semantics

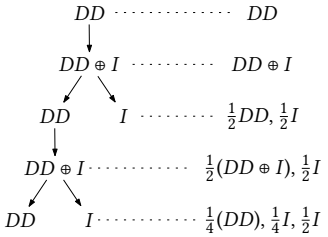
The operational semantics of $\Lambda_{\oplus}^{\mathtt{cbv}}$ is formalized through the notion of multidistribution as introduced in Section 3.1, following [Avanzini et al. 2020]. To understand why this is a convenient way

to describe the probabilistic dynamics of programs, let us consider how terms in $\Lambda_{\oplus}^{\mathsf{cbv}}$ 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,



(a) Evaluating $(I \oplus II) \oplus II$



(b) Evaluating DD

Fig. 4. Evaluating some Terms in $\Lambda_{\oplus}^{\mathsf{cbv}}$

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

- We first define a *reduction relation* $\rightarrow$ from terms to multidistributions, e.g., $M \oplus N \rightarrow \left\langle \frac{1}{2}M, \frac{1}{2}N \right\rangle$. The one-step reduction $\rightarrow \subseteq \Lambda_{\oplus}^{\mathsf{cbv}} \times \mathcal{M}(\Lambda_{\oplus}^{\mathsf{cbv}})$ is defined in Figure 5. A term M is *normal*, or in *normal form*, if there is no $\mathtt{m}$ such that $M \rightarrow \mathtt{m}$. 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* $\mathtt{m}$ such that $M \rightarrow \mathtt{m}$.
- Then we lift reduction of terms to *reduction of multidistributions* in the natural way, obtaining $\rightrightarrows$, e.g., $\left\langle \frac{1}{2}II, \frac{1}{2}(M \oplus N) \right\rangle \rightrightarrows \left\langle \frac{1}{2}I, \frac{1}{4}M, \frac{1}{4}N \right\rangle$. The relation $\rightarrow \subseteq \Lambda_{\oplus}^{\mathsf{cbv}} \times \mathcal{M}(\Lambda_{\oplus}^{\mathsf{cbv}})$ lifts to a relation $\rightrightarrows \subseteq \mathcal{M}(\Lambda_{\oplus}^{\mathsf{cbv}}) \times \mathcal{M}(\Lambda_{\oplus}^{\mathsf{cbv}})$ as defined in Figure 6. The way $\rightrightarrows$ is defined implies that *all* reducible terms in the underlying multidistributions are actually reduced according to $\rightarrow$.

[4]In the spirit Markov Decision Processes, see *e.g.* [Puterman 1994].

$$\frac{}{(\lambda x.M)V \to \langle M\{V/x\}\rangle}\ \beta \qquad \frac{}{\texttt{let } x = V \texttt{ in } M \to \langle M\{V/x\}\rangle}\ \texttt{let}V$$

$$\frac{}{M \oplus N \to \left\langle \tfrac{1}{2}M, \tfrac{1}{2}N \right\rangle}\ \oplus \qquad \frac{N \to \langle p_i N_i\rangle_{i\in I}}{(\texttt{let } x = N \texttt{ in } M) \to \langle p_i(\texttt{let } x = N_i \texttt{ in } M)\rangle}\ \texttt{let}C$$

Fig. 5. The One-step Reduction Relation $\to$

$$\frac{}{\langle V\rangle \rightrightarrows \langle V\rangle} \qquad \frac{M \to \mathtt{m}}{\langle M\rangle \rightrightarrows \mathtt{m}} \qquad \frac{(\langle M_i\rangle \rightrightarrows \mathtt{m}_i)_{i\in I}}{\langle p_i M_i \mid i \in I\rangle \rightrightarrows \biguplus_{i\in I} p_i \cdot \mathtt{m}_i}$$

Fig. 6. The Lifting of $\to$

Reduction Sequences. A $\rightrightarrows$-sequence (or *reduction sequence*) from $\mathtt{m}$ is a sequence $\mathtt{m} = \mathtt{m}_0, \mathtt{m}_1, \mathtt{m}_2, \ldots$ such that $\mathtt{m}_i \rightrightarrows \mathtt{m}_{i+1}$ for every i. Notice that since multidistribution reduction is deterministic, each $\mathtt{m}_0$ has a unique maximal reduction sequence, which is infinite and which we write $\{\mathtt{m}_n\}_{n\in\mathbb{N}}$. We write $\mathtt{m}_0 \rightrightarrows^* \mathtt{m}$ to indicate the existence of a finite reduction sequence from $\mathtt{m}_0$, and $\mathtt{m}_0 \rightrightarrows^k \mathtt{m}$ to specify the number k of $\rightrightarrows$-steps. Given a term M and $\mathtt{m}_0 = \langle M\rangle$, the sequence $\mathtt{m}_0 \rightrightarrows \mathtt{m}_1 \rightrightarrows \cdots$ naturally models the evaluation of M; each $\mathtt{m}_k$ expresses the "expected" state of the system after k steps.

Example 3. *The term DD from Example 1 evaluates as follows:*

$$\begin{aligned}\langle DD\rangle &\rightrightarrows \langle DD \oplus I\rangle \rightrightarrows \left\langle \frac{1}{2}DD, \frac{1}{2}I\right\rangle \\ &\rightrightarrows \left\langle \frac{1}{2}DD \oplus I, \frac{1}{2}I\right\rangle \rightrightarrows \left\langle \frac{1}{4}DD, \frac{1}{4}I, \frac{1}{2}I\right\rangle \\ &\rightrightarrows \left\langle \frac{1}{4}DD \oplus I, \frac{1}{4}I, \frac{1}{2}I\right\rangle \rightrightarrows \left\langle \frac{1}{8}DD, \frac{1}{8}I, \frac{1}{4}I, \frac{1}{2}I\right\rangle \rightrightarrows \cdots\end{aligned}$$

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

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

$$\begin{aligned}\langle CC\rangle &\rightrightarrows \left\langle \mathbf{succ}(CC) \oplus \overline{0}\right\rangle \rightrightarrows \left\langle \frac{1}{2}\mathbf{succ}(CC), \frac{1}{2}\overline{0}\right\rangle \\ &\rightrightarrows \left\langle \frac{1}{2}\mathbf{succ}\Big(\mathbf{succ}(CC) \oplus \overline{0}\Big), \frac{1}{2}\overline{0}\right\rangle \rightrightarrows \left\langle \frac{1}{4}\mathbf{succ}\Big(\mathbf{succ}(CC)\Big), \frac{1}{4}\mathbf{succ}(\overline{0}), \frac{1}{2}\overline{0}\right\rangle \rightrightarrows \cdots\end{aligned}$$

where $\mathbf{succ}^n(\overline{0}) \rightrightarrows^{ns} \overline{n}$. Observe that the evaluation of CC is similar to that of DD. However, while in Example 3 the term I is a value, $\mathbf{succ}^n(\overline{0})$ is not. It still has to perform ns steps (where s is a constant, see Example 2) in order to reduce to the value $\overline{n}$.

Values and Multidistributions. Given a multidistribution $\mathtt{m} \in \mathcal{M}(\Lambda_\oplus)$, we indicate by $\mathtt{m}^{\mathcal{V}}$ its restriction to values. Hence if $\mathtt{m} = \langle p_i M_i\rangle_{i\in I}$, then $\|\mathtt{m}^{\mathcal{V}}\| := \sum_{M_i\in\mathcal{V}} p_i$. The real number $\|\mathtt{m}^{\mathcal{V}}\|$ is thus the probability that $\mathtt{m}$ 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 $\Lambda_\oplus^{\mathtt{cbv}}$

Let M be a closed term, and $\langle M\rangle = \mathtt{m}_0 \rightrightarrows \mathtt{m}_1 \rightrightarrows \mathtt{m}_2 \rightrightarrows \cdots$ the reduction sequence which models its evaluation. We write $\mathtt{PTerm}_k\Big(M\Big)$ for $\|\mathtt{m}_k^{\mathcal{V}}\|$, 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 $\mathtt{PTerm}\big(M\big) := \sup_n\{\mathtt{PTerm}_n\big(M\big)\}$. As an example $\mathtt{PTerm}\big(DD\big)$ 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 [Avanzini et al. 2020; Fioriti and Hermanns 2015; Kaminski et al. 2019]. As pointed out in [Fioriti and Hermanns 2015] the expected runtime can be expressed[5] in a very convenient form as, informally,

$$\begin{aligned}\mathtt{ETime}\big(M\big) &= \sum_{k\geq 0} \Pr[\text{“}M \text{ runs more than } k \text{ steps "}] \\ &= \sum_{k\geq 0} (1 - \Pr[\text{“}M \text{ terminates within } k \text{ steps "}]).\end{aligned}$$

Within our setting, the above is easily formalised as follows:

$$\mathtt{ETime}\big(M\big) = \sum_{k\geq 0} \Big(1 - \mathtt{PTerm}_k\big(M\big)\Big)\,.$$

This formulation admits a very intuitive interpretation: given the reduction sequence $\langle M\rangle = \mathtt{m}_0 \rightrightarrows \mathtt{m}_1 \rightrightarrows \mathtt{m}_2 \rightrightarrows \cdots$, each tick in time (*i.e.* each $\rightrightarrows$ 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 $\mathtt{m}_i$ reduces with probability $1 - \|\mathtt{m}_i^{\mathcal{V}}\|$.

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

$$\mathtt{ETime}_n\big(M\big) := \sum_{k=0}^{n-1} \Big(1 - \mathtt{PTerm}_k\big(M\big)\Big)\,.$$

Clearly $\mathtt{ETime}\big(M\big) = \sup_n \Big\{\mathtt{ETime}_n\big(M\big)\Big\}$.

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

$$\begin{aligned}\langle DD\rangle &\overset{1}{\rightrightarrows} \langle DD \oplus I\rangle \overset{1}{\rightrightarrows} \left\langle \frac{1}{2}DD, \frac{1}{2}I\right\rangle \overset{\frac{1}{2}}{\rightrightarrows} \left\langle \frac{1}{2}DD \oplus I, \frac{1}{2}I\right\rangle \overset{\frac{1}{2}}{\rightrightarrows} \left\langle \frac{1}{4}DD, \frac{1}{4}I, \frac{1}{2}I\right\rangle \\ &\overset{\frac{1}{4}}{\rightrightarrows} \left\langle \frac{1}{4}DD \oplus I, \frac{1}{4}I, \frac{1}{2}I\right\rangle \overset{\frac{1}{4}}{\rightrightarrows} \left\langle \frac{1}{8}DD, \frac{1}{8}I, \frac{1}{4}I, \frac{1}{2}I\right\rangle \rightrightarrows \cdots\end{aligned}$$

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

(Positive) Almost-Sure Termination in $\Lambda_{\oplus}^{\mathtt{cbv}}$. 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 $\mathtt{PTerm}\big(M\big) = 1$. Furthermore, we say that M is *expected to terminate*, or *positively almost-surely terminating* (PAST) if $\mathtt{ETime}\big(M\big)$ is finite.

[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.* [Brémaud 2017], page 27. The equivalence with the formulations we give below is spelled out in [Avanzini et al. 2020].

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,* $\mathtt{ETime}\big(M\big) < \infty$ *implies that* $\mathtt{PTerm}\big(M\big) = 1$. *Indeed,* $\sum_{i\geq 1}(1 - \mathtt{PTerm}_i\big(M\big)) < \infty$ *implies* $\lim_{i\to\infty}\big(1 - \mathtt{PTerm}_i\big(M\big)\big) = 0$, *hence* $\lim_{i\to\infty}\mathtt{PTerm}_i\big(M\big) = 1$.

However, a program may be AST, and still have infinite expected runtime. The paradigmatic example of this is a fair random walk [Billingsley 1979]. 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 AST and PAST.

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* $q.\mathtt{A}$, where $\mathtt{A}$ is an arrow type, and $q \in (0, 1] \cap \mathbb{Q}$. 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 type.
- Finally, a term M cannot in general be typed "with certitude" namely by a single intersection type $\mathcal{A}$, but rather with a *multidistribution* of intersection types $\langle p_1\mathcal{A}_1, \ldots, p_k\mathcal{A}_k\rangle$. 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.

$$\frac{}{x : \mathcal{A} \overset{0}{\vdash} x : \mathcal{A}}\ \textsc{Var} \qquad \frac{}{\overset{0}{\vdash} M : \mathbf{0}}\ \textsc{Zero}$$

$$\frac{\Gamma \overset{w}{\vdash} V : [\mathcal{A} \to \mathtt{b}] \quad \Delta \overset{v}{\vdash} W : \mathcal{A}}{\Gamma \uplus \Delta \overset{w+v}{\vdash} VW : \mathtt{b}}\ @ \qquad \frac{\Gamma \overset{w}{\vdash} M : \mathtt{a} \quad \Delta \overset{v}{\vdash} N : \mathtt{b}}{\frac{1}{2}.\Gamma \uplus \frac{1}{2}.\Delta \overset{\frac{1}{2}w+\frac{1}{2}v+1}{\vdash} M \oplus N : \frac{1}{2}\mathtt{a} \boxplus \frac{1}{2}\mathtt{b}}\ \oplus$$

$$\frac{\Gamma, x : \mathcal{A} \overset{w}{\vdash} M : \mathtt{b}}{\Gamma \overset{w+1}{\vdash} \lambda x.M : \mathcal{A} \to \mathtt{b}}\ \lambda \qquad \frac{\Gamma \overset{v}{\vdash} N : \langle p_k \mathcal{A}_k \rangle_{k \in K} \quad (\Delta_k, x : \mathcal{A}_k \overset{w_k}{\vdash} M : \mathtt{b}_k)_{k \in K}}{\Gamma \uplus_k p_k.\Delta_k \overset{v+\sum_k p_k w_k+1}{\vdash} \mathtt{let}\ x = N\ \mathtt{in}\ M : \boxplus_k\ p_k \mathtt{b}_k}\ \mathtt{let}$$

$$\frac{\Gamma \overset{w}{\vdash} V : \mathcal{A}}{\Gamma \overset{w}{\vdash} V : \langle \mathcal{A} \rangle}\ \textsc{Val} \qquad \frac{(\Gamma_i \overset{w_i}{\vdash} V : \mathtt{A}_i)_{i \in I} \qquad (q_i)_{i \in I}\ \text{scale factors}}{\uplus_i (q_i.\Gamma_i) \overset{\sum_i q_i w_i}{\vdash} V : [q_i.\mathtt{A}_i]_{i \in I}}\ !$$

Fig. 7. Non-Idempotent Intersection Type Rules for $\Lambda^{\mathtt{cbv}}_{\oplus}$

4.1 The Type System, Formally

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

$$\begin{array}{rll} \mathtt{A}, \mathtt{B} & ::= \mathcal{A} \to \mathtt{a} & \textbf{Arrow Types} \\ \mathcal{A}, \mathcal{B} & ::= [q_1.\mathtt{A}_1, ..., q_n.\mathtt{A}_n], n \geq 0 & \textbf{Intersection Types} \\ \mathtt{a}, \mathtt{b} & ::= \langle p_1 \mathcal{A}_1, ..., p_n \mathcal{A}_n \rangle, n \geq 0 & \textbf{Type Distributions} \end{array}$$

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

It is useful to notice that an intersection type is *not* a multidistribution, because the sum of the q_i such that $\mathcal{A} = [q_i.\mathtt{A}_i]_{i \in I}$ 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, e.g., $\Delta\Delta$ can have a type of the form $[1.\mathtt{A}, 1.[\mathtt{A}] \to \mathtt{a}]$. 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* Γ 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 Γ is the set $dom(\Gamma) := \{x \mid \Gamma(x) \neq []\}$. The typing context Γ is empty if $dom(\Gamma) = \emptyset$. Multiset union $\uplus$ is extended to typing contexts pointwise, i.e. $(\Gamma \uplus \Delta)(x) := \Gamma(x) \uplus \Delta(x)$, for each variable x. A typing context Γ is denoted as $[x_1 : \mathcal{A}_1, ..., x_n : \mathcal{A}_n]$ if $dom(\Gamma) \subseteq \{x_1, ..., x_n\}$ and $\Gamma(x_i) = \mathcal{A}_i$ for all $1 \leq i \leq n$. Given two typing contexts Γ and Δ such that $dom(\Gamma) \cap dom(\Delta) = \emptyset$, the typing context Γ, Δ is defined as $(\Gamma, \Delta)(x) := \Gamma(x)$ if $x \in dom(\Gamma)$, $(\Gamma, \Delta)(x) := \Delta(x)$ if $x \in dom(\Delta)$, and $(\Gamma, \Delta)(x) := []$ otherwise. Observe that $\Gamma, x : []$ is equal to Γ. If $\Gamma = x_1 : \mathcal{A}_1, \ldots, x_n : \mathcal{A}_n$, we write $q.\Gamma$ for $[x_1 : q.\mathcal{A}_1, \ldots, x_n : q.\mathcal{A}_n]$.

Typing rules. The type assignment system in Figure 7 proves judgments of the shape $\Gamma \overset{w}{\vdash} M : \mathbf{T}$, where Γ is a type context, M is a term, $w \in \mathbb{Q}$ is a weight, and $\mathbf{T}$ is a type in one of the three forms, i.e. $\mathbf{T} ::= \mathtt{A} \mid \mathtt{a} \mid \mathcal{A}$. If Π is a formal derivation of $\Gamma \overset{w}{\vdash} M : \mathbf{T}$, then w is said to be the *weight* of Π. Please notice in Figure 7 the use of the notation $q.\Gamma$ defined above.

4.2 Some Comments on the Typing Rules

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 Var-rule and the Zero-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 $\mathbf{0}$ to *any term* M. So for example, $\Delta\Delta$ is typed as $\mathbf{0}$. The purpose of 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 $\Lambda^{\mathtt{cbv}}_{\oplus}$ includes, namely applications, probabilistic sums, abstractions, and `lets`. The following discusses each of them:

- The λ 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 β-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 `let` rule serves to type the `let` construct, and is probably the most complex one. In particular, the argument M needs to be typed multiple times, one for each scaled multiset in the multidistribution $\langle p_k \mathcal{A}_k \rangle$, 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 Δ_k, weight w_k, and type $\mathtt{b}_k$) are scaled by a factor p_k. Moreover, the weight is further increased by 1, to record a `let`V-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 β-step is recorded by the λ-rule.

The last two rules, namely 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* `let`V*-step. Note however that if* N *is a non-terminating term, it never becomes a value, and therefore the* `let`V*-step never happens. Are we counting* too much *here? Obviously not, because if* N *never become a value, then any reduction sequence from* `let` $x = M$ `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 analyse in Section 6. Let us now take a look at a few degenerate cases of the typing rules:

(1) In the λ rule, $\mathcal{A}$ 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 Γ, since if $x \notin dom(\Gamma)$, then $\Gamma, x : [] $ is equal to Γ.

(2) In the !-rule, I can be empty, and the following rule is thus a derived rule:

$$\frac{}{\Gamma \overset{0}{\vdash} \lambda x.M : []}$$

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

$$\frac{\Gamma \overset{w}{\vdash} N : \mathbf{0}}{\Gamma \overset{w+1}{\vdash} \texttt{let}\ x = N\ \texttt{in}\ M : \mathbf{0}}$$

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* $\Gamma \vdash M : \mathtt{a}$. *Then* $dom(\Gamma) \subseteq fv(M)$, *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) $\overset{w}{\vdash} V : \mathcal{A}$ *and* $\mathcal{A} = \uplus_{i \in I} \mathcal{A}_i$;
(2) $\overset{w_i}{\vdash} V : \mathcal{A}_i$ *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* $\overset{qv}{\vdash} V : q.\mathcal{A}$ *iff* $\overset{v}{\vdash} V : \mathcal{A}$.

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. And as 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 $\sum_{k=1}^{\infty} \frac{1}{k^2}$ and $\sum_{k=1}^{\infty} \frac{1}{k}$: the first converges, while the second diverges.

We thus need to count steps neither "too much" nor "too little". Two crucial features make this possible: the arrows in an intersection type are *scaled* by a factor q, and we allow type derivations also for terms which receive the *null type* $\mathbf{0}$, such as $\Delta\Delta$. The first feature allows us not to count "too much", the second not to count "too little". In this section, we illustrate these aspects by way of some concrete examples.

5.1 Not Too Much

In intersection type systems for the λ-calculus such as those by Lengrand and co-authors [Accattoli et al. 2018, 2019; Bernadet and Lengrand 2013], the weight of any type derivation accounts for how many times redexes can be fired. Roughly speaking, to each λ-abstraction in the type derivation corresponds a β-redex being fired, therefore to measure the runtime of a λ-term, the weight is increased by one at each instance of the λ 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 λ rule as in the deterministic case is simply too much. Consider our running example DD, where $D = \lambda x.xx \oplus I$, and $\mathtt{ETime}\big(DD\big) = 4$. It is easy to see that for each $k \in \mathbb{N}$, there is a derivation Π which contains k instances of λ rule. If each is counted 1, we would have a derivation $\Pi \overset{k}{\vdash} DD : \mathtt{a}$ for every k, and so $\sup\{k \mid \overset{k}{\vdash} DD : \mathtt{a}\} = \infty$. Instead, we need to scale each instance of λ (say, with conclusion $\mathtt{A}$) by *the probability p of the λ abstraction to be involved in a redex*. Such an information is stored as a scalar somewhere else in the derivation.

To clarify, let us examine our running example. We want to capture $\mathtt{ETime}_n\big(DD\big)$ and the fact that $\mathtt{PTerm}_{2n}\big(DD\big)$ is $\left(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^n}\right)$. We define the types $\mathtt{A}_n$ and $\mathcal{A}_n$ as follows.

$$\mathcal{A}_0 = [] \qquad \mathtt{A}_n = \mathcal{A}_{n-1} \to \Big\lfloor\!\!\Big\lfloor \Big\langle \frac{1}{2^k}[] \Big\rangle \Big\rfloor\!\!\Big\rfloor_{k=1}^{n} \qquad \mathcal{A}_n = \frac{1}{2}.\mathcal{A}_{n-1} \uplus \frac{1}{2}.[\mathtt{A}_n]$$

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

$$\mathtt{A}_1 = [] \to \Big\langle \frac{1}{2}[] \Big\rangle, \qquad \mathtt{A}_2 = \mathcal{A}_1 \to \Big\langle \frac{1}{4}[], \frac{1}{2}[] \Big\rangle, \qquad \mathtt{A}_3 = \mathcal{A}_2 \to \Big\langle \frac{1}{8}[], \frac{1}{4}[], \frac{1}{2}[] \Big\rangle$$
$$\mathcal{A}_1 = \Big[\frac{1}{2}.\mathtt{A}_1\Big], \qquad \mathcal{A}_2 = \Big[\frac{1}{4}.\mathtt{A}_1, \frac{1}{2}.\mathtt{A}_2\Big], \qquad \mathcal{A}_3 = \Big[\frac{1}{8}.\mathtt{A}_1, \frac{1}{4}.\mathtt{A}_2, \frac{1}{2}.\mathtt{A}_3\Big]$$

The value D can be given all the arrow types $\mathtt{A}_i$, for every i, all these derivations having weight equal to 2, i.e. for every $i \geq 1$ there is a derivation Σ_i such that $\Sigma_i \triangleright \overset{2}{\vdash} \lambda x.xx \oplus I : \mathtt{A}_i$. For example, the type derivations Σ_1 to Σ_3 can be built as follows:

$$\Sigma_1 \triangleright \dfrac{\dfrac{\overset{0}{\vdash} I : \langle[]\rangle}{\overset{1}{\vdash} xx \oplus I : \Big\langle \frac{1}{2}[] \Big\rangle}}{\overset{2}{\vdash} \lambda x.xx \oplus I : [] \to \Big\langle \frac{1}{2}[] \Big\rangle} \qquad \Sigma_2 \triangleright \dfrac{\dfrac{x : [\mathtt{A}_1] \overset{0}{\vdash} x : [[] \to \Big\langle \frac{1}{2}[] \Big\rangle] \quad x : [] \overset{0}{\vdash} x : []}{x : [\mathtt{A}_1] \overset{0}{\vdash} xx : \Big\langle \frac{1}{2}[] \Big\rangle} \qquad \overset{0}{\vdash} I : \langle[]\rangle}{\dfrac{x : [\frac{1}{2}.\mathtt{A}_1] \overset{1}{\vdash} xx \oplus I : \Big\langle \frac{1}{4}[], \frac{1}{2}[] \Big\rangle}{\overset{2}{\vdash} \lambda x.xx \oplus I : [\frac{1}{2}.\mathtt{A}_1] \to \Big\langle \frac{1}{4}[], \frac{1}{2}[] \Big\rangle}}$$

$$\Sigma_3 \triangleright \dfrac{\dfrac{\dfrac{x : [\mathtt{A}_2] \overset{0}{\vdash} x : [[\frac{1}{2}.\mathtt{A}_1] \to \Big\langle \frac{1}{4}[], \frac{1}{2}[] \Big\rangle] \quad x : [\frac{1}{2}.\mathtt{A}_1] \overset{0}{\vdash} x : [\frac{1}{2}.\mathtt{A}]}{x : [\mathtt{A}_2, \frac{1}{2}.\mathtt{A}_1] \overset{0}{\vdash} xx : \Big\langle \frac{1}{4}[], \frac{1}{2}[] \Big\rangle} \qquad \dfrac{}{\overset{0}{\vdash} I : \langle[]\rangle}}{x : [\frac{1}{2}.\mathtt{A}_2, \frac{1}{4}.\mathtt{A}_1] \overset{1}{\vdash} xx \oplus I : \Big\langle \frac{1}{8}[], \frac{1}{4}[], \frac{1}{2}[] \Big\rangle}}{\overset{2}{\vdash} \lambda x.xx \oplus I : [\frac{1}{2}.\mathtt{A}_2, \frac{1}{4}.\mathtt{A}_1] \to \Big\langle \frac{1}{8}[], \frac{1}{4}[], \frac{1}{2}[] \Big\rangle}$$

One can attribute to D also any intersection type $\mathcal{A}_j$, by collecting and scaling the j derivations $(\Sigma_j, \ldots, \Sigma_1)$ by way of the rule ! (with scale factors $\frac{1}{2}, \ldots, \frac{1}{2^j}$), thus obtaining the type derivation Θ_j, this time with weight $2(\sum_{k:1}^{j} \frac{1}{2^k}) = 2 - \frac{1}{2^{j-1}}$. Finally, Σ_{j+1} and Θ_j can be aggregated in the

derivation Φ_{j+1}, typing DD. Note that the weight is now $2(\sum_{k:0}^{j} \frac{1}{2^k})$.

$$\dfrac{\Sigma_{j+1} \triangleright \overset{2}{\vdash} D : \mathcal{A}_j \to \biguplus_{k:1}^{j+1} \left\langle \frac{1}{2^k}[] \right\rangle \quad \Theta_j \triangleright \overset{2(\sum_{k:1}^{j} \frac{1}{2^k})}{\vdash} D : \mathcal{A}_j}{\Phi_{j+1} \triangleright \overset{2(\sum_{k:0}^{j} \frac{1}{2^k})}{\vdash} DD : \biguplus_{k:1}^{j+1} \left\langle \frac{1}{2^k}[] \right\rangle}$$

For example, we have the following derivation, which indeed corresponds to the $(2 \cdot 3)$-approximant of $\mathtt{ETime}\big(DD\big)$: recall from Example 5 that $\mathtt{ETime}_6\big(DD\big) = 3 + \frac{1}{2}$ (and that $\mathtt{PTerm}_6\big(DD\big) = \frac{7}{8}$).

$$\dfrac{\Sigma_3 \triangleright \overset{2}{\vdash} \lambda x.xx \oplus I : [\tfrac{1}{2}.\mathtt{A}_2, \tfrac{1}{4}.\mathtt{A}_1] \to \left\langle \frac{1}{8}[], \frac{1}{4}[], \frac{1}{2}[] \right\rangle \qquad \dfrac{\Sigma_2 \overset{2}{\vdash} D : \mathtt{A}_2 \quad \Sigma_1 \triangleright \overset{2}{\vdash} D : \mathtt{A}_1}{\overset{1+\frac{1}{2}}{\vdash} D : [\tfrac{1}{2}.\mathtt{A}_2, \tfrac{1}{4}.\mathtt{A}_1]}\ !}{\Phi_3 \triangleright \overset{3+\frac{1}{2}}{\vdash} DD : \left\langle \frac{1}{8}[], \frac{1}{4}[], \frac{1}{2}[] \right\rangle}$$

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:

$$\langle I \oplus \Delta\Delta \rangle \overset{1}{\rightrightarrows} \left\langle \frac{1}{2}I, \frac{1}{2}\Delta\Delta \right\rangle \overset{\frac{1}{2}}{\rightrightarrows} \left\langle \frac{1}{2}I, \frac{1}{2}\Delta\Delta \right\rangle \overset{\frac{1}{2}}{\rightrightarrows} \left\langle \frac{1}{2}I, \frac{1}{2}\Delta\Delta \right\rangle \ldots$$

Clearly, $\mathtt{ETime}\big(M\big) = \infty$. 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 $\overset{n}{\vdash} M : \mathbf{0}$, 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 $\mathtt{A}_{i+1} = [\mathtt{A}_1, \ldots, \mathtt{A}_i] \to \mathbf{0}$ (so, in particular, $\mathtt{A}_1 = [] \to \mathbf{0}$, $\mathtt{A}_2 = [\mathtt{A}_1] \to \mathbf{0}$). For each i, one can build a derivation Σ_i having weight 1 and typing Δ with $\mathtt{A}_i$. Here are a couple of examples:

$$\Sigma_1 \triangleright \dfrac{\overset{0}{\vdash} xx : \mathbf{0}}{\overset{1}{\vdash} \lambda x.xx : [] \to \mathbf{0}} \qquad \Sigma_2 \triangleright \dfrac{\dfrac{x : [[] \to \mathbf{0}] \overset{0}{\vdash} x : [] \to \mathbf{0}] \quad \overset{0}{\vdash} x : []}{x : [[] \to \mathbf{0}] \overset{0}{\vdash} xx : \mathbf{0}}}{\overset{1}{\vdash} \lambda x.xx : [[] \to \mathbf{0}] \to \mathbf{0}}$$

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

$$\dfrac{\Sigma_3 \triangleright \overset{1}{\vdash} \lambda x.xx : [\,[[] \to \mathbf{0}] \to \mathbf{0},\ [] \to \mathbf{0}\,] \to \mathbf{0} \qquad \Sigma_2 \triangleright \overset{1}{\vdash} \lambda x.xx : [[] \to \mathbf{0}] \to \mathbf{0} \qquad \Sigma_1 \triangleright \overset{1}{\vdash} \lambda x.xx : [] \to \mathbf{0}}{\overset{3}{\vdash} (\lambda x.xx)\lambda x.xx : \mathbf{0}}\ @$$

6 CHARACTERISING 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 and 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, $\mathtt{ETime}\big(M\big)$ (respectively, $\mathtt{PTerm}\big(M\big)$) bounds from above the weight w (respectively, the norm $\|\mathtt{a}\|$) of any type derivation $\Pi \triangleright \overset{w}{\vdash} M : \mathtt{a}$. 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 $\mathtt{ETime}\big(I\big) = 0$. It can be given the type $\mathbf{0}$ (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*

$$\dfrac{\dfrac{}{x : [] \overset{0}{\vdash} x : []}\ \textsc{Var}}{\overset{1}{\vdash} \lambda x.x : [] \to []}\ \lambda$$

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 $[] \to \mathbf{0}$, (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 $\mathtt{a}$ is said to be *tight* if it is a multidistribution on the empty intersection type $[]$. Accordingly, a derivation $\Pi \triangleright \overset{w}{\vdash} M : \mathtt{a}$ is said to be *tight* if $\mathtt{a}$ is tight.

A tight type has therefore shape $\mathtt{a} = \langle q_k[]\rangle_{k\in K}$, where K is possibly empty. In particular the null type $\mathbf{0}$ is a tight type. Observe that if $\mathtt{a}$ is tight, then $\|\mathtt{a}\| = \sum_k q_k$ (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:*

$$\dfrac{\dfrac{}{\overset{0}{\vdash} V : []}\ !}{\overset{0}{\vdash} V : \langle 1[]\rangle}\ \textsc{Val} \qquad \dfrac{}{\overset{0}{\vdash} V : \mathbf{0}}\ \textsc{Zero}$$

Looking back at Example 12, one immediately realises 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 $\mathtt{PTerm}_k\big(M\big) \le \mathtt{PTerm}_{k+1}\big(M\big)$ and $\mathtt{ETime}_k\big(M\big) \le \mathtt{ETime}_{k+1}\big(M\big)$. Moreover, if $M \to \langle q_i M_i\rangle_{i\in I}$ then*

$$\mathtt{PTerm}_{k+1}\big(M\big) = \sum_{i\in I} q_i \left(\mathtt{PTerm}_k\big(M_i\big)\right), \qquad \mathtt{ETime}_{k+1}\big(M\big) = 1 + \sum_{i\in I} q_i \left(\mathtt{ETime}_k\big(M_i\big)\right).$$

6.2 Soundness

In this section, we prove the correctness of our type system. Namely, we prove that if $\overset{w}{\vdash} M : \mathtt{a}$ is (tightly) derivable, then M has probability of termination at least $\|\mathtt{a}\|$, 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 Π (denoted $|\Pi|$) is the standard one, and is defined as the number of rules in Π (excluding the !-rule and the Val-rule, which cannot be iterated).

Lemma 16 (Weighted Subject Reduction). *Suppose that* $\Pi \triangleright \overset{w}{\vdash} P : \mathtt{b}$, *with* $w > 0$, *and that* $P \to \langle q_i P_i \rangle_{i\in I}$. *Then for every* $i \in I$ *there exists a derivation* Π_i *such that* $\Pi_i \triangleright \overset{w_i}{\vdash} P_i : \mathtt{b}_i$, *and* $|\Pi| > |\Pi_i|$. *Moreover,* $\mathtt{b} = \biguplus_{i\in I} q_i \mathtt{b}_i$ *and* $w = 1 + \sum_{i\in I} q_i w_i$.

The proof is in the Extended Version of this paper. 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* $\overset{w}{\vdash} M : \mathtt{b}$, *there exists* $k \in \mathbb{N}$ *such that* $\|\mathtt{b}\| \le \mathtt{PTerm}_k\left(M\right)$ *and* $w \le \mathtt{ETime}_k\left(M\right)$.

Proof. By induction on the size $|\Pi|$ of the type derivation Π such that $\Pi \triangleright \overset{w}{\vdash} M$, 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 $\mathtt{PTerm}_0\left(M\right) = \| \langle M \rangle^{\mathcal{V}} \| = 1$ and $\mathtt{ETime}_0\left(M\right) = 0$.
- 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 $\overset{0}{\vdash} M : \mathbf{0}$ which trivially satisfies the claim, with $k = 0$.
  - If $w > 0$, then since M is not normal, it has a reduction step $M \to \langle q_i M_i \rangle_{i\in I}$. By Weighted Subject Reduction (Lemma 16), we derive that for each $i \in I$ there exists a derivation $\Pi_i \triangleright \overset{w_i}{\vdash} M_i : \mathtt{b}_i$, with $|\Pi_i| < |\Pi|$. Since $\mathtt{b}$ is tight, necessarily each $\mathtt{b}_i$ also is tight, again by Lemma 16 (observe also that, if $\mathtt{b} = \mathbf{0}$, then $\mathtt{b}_i = \mathbf{0}$). By *i.h.*, for each $M_i (i \in I)$ there exists $k_i \in \mathbb{N}$ which satisfies the conditions on $\|\mathtt{b}_i\|$ and w_i. Let $h = max\{k_i\}_{i\in I}$. Since $h \ge k_i$, for each $i \in I$ we have $\|\mathtt{b}_i\| \le \mathtt{PTerm}_h\left(M_i\right)$ and $w_i \le \mathtt{ETime}_h\left(M_i\right)$. Moreover, Weighted Subject Reduction implies also that $\|\mathtt{b}\| = \sum_i q_i \|\mathtt{b}_i\|$ and $w = 1 + \sum_i q_i w_i$. The claim follows easily by Property 15, with $k = h + 1$. Indeed $\mathtt{PTerm}_{h+1}\left(M\right) = \left(\sum_{i\in I} q_i \mathtt{PTerm}_h\left(M_i\right)\right) \ge \left(\sum_{i\in I} q_i \|\mathtt{b}_i\|\right) = \|\mathtt{b}\|$ and $\mathtt{ETime}_{h+1}\left(M\right) = \left(1 + \sum_{i\in I} q_i \mathtt{ETime}_h\left(M_i\right)\right) \ge \left(1 + \sum_{i\in I} q_i w_i\right) = w$.

Since there are no other cases, we are done. □

Observe that Theorem 17 holds for every tight type $\mathtt{b}$, including the null type. Thus it has the following immediate consequence:

Corollary 18 (Finitary Soundness of Null Typing). *Let* M *be a* closed *term such that* $\overset{w}{\vdash} M : \mathbf{0}$. *Then there exists* $k \in \mathbb{N}$ *such that* $w \le \mathtt{ETime}_k\left(M\right)$.

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 Subject Expansion.

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

The proof is in the extended version of this paper [Dal Lago et al. 2020]. 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 $\Pi \triangleright \overset{w}{\vdash} M : \mathtt{a}$, such that $\|\mathtt{a}\| = \mathtt{PTerm}_k\left(M\right)$ and $w \geq \mathtt{ETime}_k\left(M\right)$.*

Proof. By induction on k, distinguishing some cases.

- If M is a value, then for each k, $\mathtt{PTerm}_k\left(M\right) = 1$ and $\mathtt{ETime}_k\left(M\right) = 0$. The derivation $\overset{0}{\vdash} M : \langle 1[] \rangle$ (Lemma 14) satisfies the claim.
- Otherwise, if M is not a value:
  - If $k = 0$, we have $\mathtt{PTerm}_0\left(M\right) = 0 = \mathtt{ETime}_0\left(M\right)$; the Zero-rule satisfies the claim.
  - If $k > 0$, assume $M \to \langle p_i M_i \rangle_{i\in I}$. By *i.h.*, for each $i \in I$, there exists a tight derivation $\Pi_i \triangleright \overset{w_i}{\vdash} M_i : \mathtt{a}_i$, such that $\|\mathtt{a}_i\| = \mathtt{PTerm}_{k-1}\left(M_i\right)$ and $w_i \geq \mathtt{ETime}_{k-1}\left(M_i\right)$. By Weighted Subject Expansion, there exists a tight derivation $\Phi \triangleright \overset{w}{\vdash} M : \mathtt{a}$ such that $\|\mathtt{a}\| = \sum_{i\in I} p_i \|\mathtt{a}_i\|$, and $w \geq \sum_{i\in I} p_i w_i$. We conclude by Property 15, because $\mathtt{PTerm}_k\left(M\right) = \sum_{i\in I} p_i \mathtt{PTerm}_{k-1}\left(M_i\right) = \sum_{i\in I} p_i \|\mathtt{a}_i\| = \|\mathtt{a}\|$ and $\mathtt{ETime}_k\left(M\right) = 1 + \sum_{i\in I} p_i \mathtt{ETime}_{k-1}\left(M_i\right) \leq 1 + \sum_{i\in I} p_i w_i \leq w$.

□

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 $\mathtt{PTerm}\left(M\right)$ and $\mathtt{ETime}\left(M\right)$ 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*

$$\mathtt{PTerm}\left(M\right) = \sup\{\|\mathtt{a}\| \mid \exists \Pi.\Pi \triangleright \vdash M : \mathtt{a} \textit{ is a tight derivation}\}$$

$$\mathtt{ETime}\left(M\right) = \sup\{w \mid \exists \Pi.\exists \mathtt{a}.\Pi \triangleright \overset{w}{\vdash} M : \mathtt{a} \textit{ is tight derivation}\}$$

Proof. Let us first of all define $\mathbf{N}(M)$ and $\mathbf{W}(M)$ as follows:

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

$$\mathbf{W}(M) := \sup\{w \mid \exists \Pi.\exists \mathtt{a}.\Pi \triangleright \overset{w}{\vdash} M : \mathtt{a} \text{ is tight derivation}\}$$

We now proceed by proving the following two statements:

- On the one hand, we prove that $\mathbf{N}(M) = \mathtt{PTerm}\left(M\right)$, recalling that we defined $\mathtt{PTerm}\left(M\right)$ as $\sup\{\mathtt{PTerm}_n\left(M\right) \mid n \in \mathbb{N}\}$.

- Let $\Phi \triangleright \vdash M : \mathtt{a}$ be a tight derivation. By Correctness (Theorem 17), $\|\mathtt{a}\| \leq \mathtt{PTerm}\Big(M\Big)$. Hence $\mathbf{N}(M) \leq \mathtt{PTerm}\Big(M\Big)$.
- By Finitary Completeness (Theorem 20), for each $k \in \mathbb{N}$ there exists a tight derivation $\Pi \triangleright \vdash M : \mathtt{a}$ such that $\mathtt{PTerm}_k\Big(M\Big) \leq \|\mathtt{a}\|$. Hence $\mathbf{N}(M) = \mathtt{PTerm}\Big(M\Big)$.

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

□

By definition, M is AST iff $\mathtt{PTerm}\Big(M\Big) = 1$, while M is PAST iff $\mathtt{ETime}\Big(M\Big)$ is finite. As a consequence:

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

Example 23. *In Section 5 we have discussed tight derivations for our running example DD. Each derivation Σ_i for D has constant weight 2. Collecting and scaling the $j+1$ derivations $\Sigma_{j+1}, \Sigma_j, \ldots, \Sigma_1$, we obtained a derivation Φ_{j+1} for DD, with weight $2(1 + \frac{1}{2} + \ldots + \frac{1}{2^j}) = 2(\sum_{n \leq j} \frac{1}{2^n}) < 4$. Let us also sketch how type derivations can be built for the terms CC and $\mathbf{exp}(CC)$ from Example 2.*

- *We can build a tight type derivation Ξ_i for $C = \Big(\lambda x.(\mathtt{let}\ z = xx\ \mathtt{in}\ SUCC\ z) \oplus \overline{0}\Big)$ by following the blueprint of the derivation Σ_i for D. The weight of each Ξ_i is $2 + \frac{1}{2}v_i$, where the weight v_i is contributed by the* let *rule, and increases as i increases, because the* let *rule has more than one premiss. By collecting and appropriately scaling the derivations $\Xi_j, \ldots, \Xi_1$, and putting this together with Ξ_{j+1}, we then obtain a type derivation Ψ_{j+1} for CC (similarly to what we have done to obtain Φ_{j+1}). The weight has now a bound similar to that for Φ_{j+1}, plus an overhead which is obtained by summing the scaled $\frac{1}{2}v_i$'s, giving an overall weight $w < 4 + s$.*
- *An even more interesting term is $\mathbf{exp}(CC)$. We can build tight derivations for it from appropriate derivations for CC. It is clear that the term $EXP\ \overline{n}$ can be given a tight derivation of weight (at least) 2^n, in a standard way. From there, for every j, a tight type derivation having weight at least $\frac{1}{2}\sum_{n \leq j} \frac{2^n}{2^n} = \frac{j+1}{2}$ can be built, so the set of tight weights is unbounded.*

6.4.2 Focusing on Expected Runtimes. Remember that PAST $\subset$ AST. The previous characterisation may give the impression that analysing the runtime of a term somehow requires studying its probability of termination. In fact, intersection types allow us to establish PAST *independently* from AST, by looking only at the type $\mathbf{0}$ 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 $\mathbf{0}$. 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 $\Pi \triangleright \overset{w}{\vdash} M : \mathbf{0}$, such that $w \geq \mathtt{ETime}_k\Big(M\Big)$.*

Proof. The proof is a simplification of Theorem 20. We only need to observe that the $\mathbf{0}$ typing is preserved by subject expansion, and the weight strictly increases along it. □

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:*

$$\mathtt{ETime}\Big(M\Big) = \sup\{w \mid \Pi \triangleright \overset{w}{\vdash} M : \mathbf{0}\} \qquad M \in \mathsf{PAST} \Leftrightarrow \sup\{w \mid \Pi \triangleright \overset{w}{\vdash} M : \mathbf{0}\} < \infty$$

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 $\Pi \triangleright \overset{w}{\vdash} DD : \mathbf{0}$ in such a way that w is bounded by $\mathtt{ETime}\Big(DD\Big) = 4$, and for each approximant $\mathtt{ETime}_k\Big(DD\Big)$ 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 $\{\mathtt{B}_n\}_{n\in\mathbb{N}}$ and $\{\mathcal{B}_n\}_{n\in\mathbb{N}}$ as follows:

$$\mathcal{B}_0 = [] \qquad \mathcal{B}_n = \frac{1}{2}.\mathcal{B}_{n-1} \uplus \frac{1}{2}.[\mathtt{B}_n] \qquad \mathtt{B}_n = \mathcal{B}_{n-1} \to \mathbf{0}$$

For example:

$$\mathtt{B}_1 = [] \to \mathbf{0}, \qquad \mathtt{B}_2 = \mathcal{B}_1 \to \mathbf{0}, \qquad \mathtt{B}_3 = \mathcal{B}_2 \to \mathbf{0},$$
$$\mathcal{B}_1 = \left[\frac{1}{2}.\mathtt{B}_1\right] \qquad \mathcal{B}_2 = \left[\frac{1}{4}.\mathtt{B}_1, \frac{1}{2}.\mathtt{B}_2\right], \qquad \mathcal{B}_3 = \left[\frac{1}{8}.\mathtt{B}_1, \frac{1}{4}.\mathtt{B}_2, \frac{1}{2}.\mathtt{B}_3\right]$$

The given types are structurally very similar to those from Section 5. We can thus mimic the constructions given there, and get derivations Σ_i, each having weight 2 and typing D with $\mathtt{B}_i$, but also derivations having weights converging to 4, this time typing DD with $\mathbf{0}$. So for example, recalling that $\mathtt{ETime}_6\Big(DD\Big) = 3 + \frac{1}{2}$, here is the corresponding type derivation:

$$\dfrac{\dfrac{\dfrac{x : [[\frac{1}{2}.\mathtt{B}_1] \to \mathbf{0}] \overset{0}{\vdash} x : [[\frac{1}{2}.\mathtt{B}_1] \to \mathbf{0}] \quad x : [\frac{1}{2}.\mathtt{B}_1] \overset{0}{\vdash} x : [\frac{1}{2}.\mathtt{B}_1]}{x : [\mathtt{B}_2, \frac{1}{2}.\mathtt{B}_1] \overset{0}{\vdash} xx : \mathbf{0}} \qquad \overset{0}{\vdash} I : \mathbf{0}}{x : [\frac{1}{2}.\mathtt{B}_2, \frac{1}{4}.\mathtt{B}_1] \overset{1}{\vdash} xx \oplus I : \mathbf{0}} \oplus}{\Sigma_3 \triangleright \overset{2}{\vdash} \lambda x.xx \oplus I : [\frac{1}{2}.\mathtt{B}_2, \frac{1}{4}.\mathtt{B}_1] \to \mathbf{0}} \qquad \dfrac{\Sigma_1 \triangleright \overset{2}{\vdash} D : \mathtt{B}_1 \quad \Sigma_2 \overset{2}{\vdash} D : \mathtt{B}_2}{\Sigma \triangleright \overset{1+\frac{1}{2}}{\vdash} D : [\frac{1}{2}.\mathtt{B}_2, \frac{1}{4}.\mathtt{B}_1]} \; !$$
$$\overset{3+\frac{1}{2}}{\vdash} DD : \mathbf{0}$$

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 conclusions of a certain form 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. [Kaminski et al. 2019], which establish that in the realm of probabilistic Turing machines, almost-sure termination is a Π^0_2-complete problem, while positive almost-sure termination is Σ^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 $\Lambda^{\mathtt{cbv}}_{\oplus}$, witnessing the fact that the aforementioned recursion-theoretic limitations also hold for $\Lambda^{\mathtt{cbv}}_{\oplus}$.
- On the other hand, we prove *by way of our type system* that the class of positively almost-surely terminating terms in $\Lambda^{\mathtt{cbv}}_{\oplus}$ is Σ^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 Π^0_2.

7.1 Probabilistic Turing Machines

Probabilistic Turing machines [Gill 1977; Santos 1969] (PTMs in the following) can be defined similarly to ordinary deterministic ones, the main difference being the fact that the transition function δ 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 $\Lambda^{\mathtt{cbv}}_{\oplus}$ make perfect sense for Turing machines too, e.g., given a probabilistic Turing machine $\mathcal{M}$ and an input $x \in \Sigma^*$, we say that $\mathcal{M}$ *is AST on* x if $\mathcal{M}$ converges with probability 1. Like ordinary Turing machines, PTMs can be effectively enumerated and the PTM corresponding to α is indicated as $\mathcal{M}_\alpha$. This allows us to introduce the following classes of (pairs of) natural numbers:

$$\mathsf{AST}_{TM} = \{(\alpha, x) \mid \text{the PTM } \mathcal{M}_\alpha \text{ is AST on input } x\}$$
$$\mathsf{PAST}_{TM} = \{(\alpha, x) \mid \text{the PTM } \mathcal{M}_\alpha \text{ is PAST on input } x\}$$

7.2 Encoding PTMs into $\Lambda_{\oplus}$

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

$$V ::= x \mid \lambda x.M \qquad \textbf{Values}, \mathcal{V}^{\mathtt{cps}}$$
$$M ::= V \mid MV \mid M \oplus M \qquad \textbf{Terms}, \Lambda^{\mathtt{cps}}_{\oplus}$$

where MV is nothing more than syntactic sugar for $\mathtt{let}\ x\ =\ M\ \mathtt{in}\ xV$. In doing so, we follow [Dal Lago and Accattoli 2017], 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* [Wadsworth 1980], e.g., given an alphabet $\Sigma = \{a_1, \ldots, a_m\}$ strings in Σ^* are encoded following the recursive definition below:
$$\overline{\varepsilon} = \lambda x_1.\cdots.\lambda x_m.\lambda y.y \qquad \overline{a_i \cdot s} = \lambda x_1.\cdots.\lambda x_m.\lambda y.x_i\overline{s}$$
- Similarly, one can encode any tuple of values $(V_1, \ldots, V_n)$ as $\lambda x.xV_1 \cdots V_n$. This encoding easily supports projections.
- We can build a fixed-point combinator Z as MM, where M is the term $\lambda x.\lambda y.y(\lambda z.xxyz)$. Observe that for every value V, it holds that ZV deterministically rewrites (in a constant amount of steps) to $V(\lambda x.ZVx)$. Notice that the argument to V is *not* ZV, but is "wrapped" into a value by way of η-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 [Dal Lago and Accattoli 2017], except in the encoding of the transition function), one can reach the following:

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

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 [Odifreddi 1989].

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

$$x \in X \Leftrightarrow \underbrace{\exists y_1.\forall y_2 \exists y_3.\forall y_4 \ldots}_{n \text{ times}} R(x, y_1, \ldots, y_n)$$

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

$$x \in X \Leftrightarrow \underbrace{\forall y_1.\exists y_2 \forall y_3.\exists y_4 \ldots}_{n \text{ times}} R(x, y_1, \ldots, y_n)$$

For both the classes Σ^0_n and Π^0_n, there are related notions of *hardness*: a set $X \subseteq \mathbb{N}$ is Σ^0_n-difficult (respectively, Π^0_n-difficult) iff it is *at least as difficult as* any other Σ^0_n (respectively, Π^0_n) problem, i.e. if for every other Π^0_n problem Y there is a (recursive) reduction from Y to X. Both in Σ^0_n and in Π^0_n, *completeness* stands for containment *and* hardness. These classes form an hierarchy which is strict; moreover, Σ^0_n and Π^0_n, although having non-empty intersections, are incomparable as classes.

Where, in the arithmetical hierarchy, do AST_{TM} and AST_{TM} reside? A precise answer to this question has been given by Kaminski et al. [Kaminski et al. 2019] in the realm of while programs, but can easily be rephrased for PTMs:

Theorem 27 (Kaminski et al. [Kaminski et al. 2019]). AST_{TM} *is Π^0_2-complete, while* PAST_{TM} *is Σ^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 λ-calculus) terminating computations form a Σ^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 λ-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 Σ^0_1.

7.4 The Optimality Result

In the probabilistic λ-calculus, neither form of termination is Σ^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 λ-counterparts of AST_{TM} and PAST_{TM}:

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

Theorem 26 and Theorem 27 together imply that AST_λ is Π^0_2-hard and PAST_λ is Σ^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_λ is *in* Π^0_2 and that PAST_λ is *in* Σ^0_2. Indeed, consider the following two sets

$$\mathsf{AST}_{\lambda,\vdash} = \{M \mid \forall r \in \mathbb{Q}_{[0,1)}.\exists \Pi.(\Pi \triangleright \vdash M : \mathsf{a}) \wedge (\|\mathsf{a}\| > r)\};$$

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

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

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 λ-calculus equipped with weak head reduction, itself already studied in many papers from the literature [Dal Lago et al. 2014; Dal Lago and Zorzi 2012]. We first define the language, called $\Lambda^{\mathtt{cbn}}_\oplus$, and its operational semantics, then the typing system.

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

$$V ::= x \mid \lambda x.M \qquad \textbf{Values}, \mathcal{V}^{\mathtt{cbn}}_\oplus$$

$$M ::= V \mid MM \mid M \oplus M \qquad \textbf{Terms}, \Lambda^{\mathtt{cbn}}_\oplus$$

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

$$\frac{}{(\lambda x.M)V \to \langle M\{V/x\}\rangle}\ \beta \qquad \frac{N \to \langle p_i N_i\rangle_{i\in I}}{NM \to \langle p_i (N_i M)\rangle_{i\in I}}\ \texttt{head}$$

Fig. 8. Reduction Steps

$$\frac{}{x : [1.\mathtt{a}] \overset{0}{\vdash} x : \mathtt{a}}\ \textsc{Var} \qquad \frac{}{\overset{0}{\vdash} \lambda x.M : \langle * \rangle}\ \textsc{Val} \qquad \frac{}{\overset{0}{\vdash} M : \mathbf{0}}\ \textsc{Zero}$$

$$\frac{\Gamma, x : \mathcal{A} \overset{w}{\vdash} M : \mathtt{b}}{\Gamma \overset{w+1}{\vdash} \lambda x.M : \langle \mathcal{A} \to \mathtt{b}\rangle}\ \lambda \qquad \frac{\Gamma \overset{w}{\vdash} M : \langle p_k(\mathcal{A}_k \to \mathtt{b}_k)\rangle_{k\in K} \quad \left(\Pi_k \triangleright \Delta_k \overset{w_k}{\vdash} N : \mathcal{A}_k\right)_{k\in K}}{\Gamma \uplus_k p_k.\Delta_k \overset{w+\sum_k p_k w_k}{\vdash} MN : \lfloor + \rfloor_k\ p_k \mathtt{b}_k}\ @$$

$$\frac{(\Gamma_i \overset{w_i}{\vdash} M : \mathtt{a}_i)_{i\in I} \qquad (q_i)_{i\in I} \text{ scale factors}}{\uplus_i (q_i.\Gamma_i) \overset{\sum_i q_i w_i}{\vdash} M : [q_i.\mathtt{a}_i]_{i\in I}}\ ! \qquad \frac{\Gamma \overset{w_1}{\vdash} M : \mathtt{a} \quad \Delta \overset{w_2}{\vdash} N : \mathtt{b}}{\frac{1}{2}.\Gamma \uplus \frac{1}{2}.\Delta \overset{1+\frac{1}{2}w_1+\frac{1}{2}w_2}{\vdash} M \oplus N : \frac{1}{2}\mathtt{a} \lfloor + \rfloor \frac{1}{2}\mathtt{b}}\ \oplus$$

Fig. 9. Non-Idempotent Intersection Type Rules for $\Lambda^{\mathtt{cbn}}_{\oplus}$

Operational Semantics and Probabilistic Termination. As in CbV, we first define a one-step reduction relation $\to$ from terms to multidistributions. The rules are given in Figure 8. We then lift $\to$ to a *reduction of multidistributions*, and this can be done as for $\Lambda^{\mathtt{cbv}}_{\oplus}$, so following the rules in Figure 6. Values are precisely the closed terms which cannot be further reduced. The definitions of $\|\mathbb{m}^{\mathcal{V}}_k\|$, $\mathtt{PTerm}_k\left(M\right)$, $\mathtt{PTerm}\left(M\right)$, $\mathtt{ETime}_k\left(M\right)$, and $\mathtt{ETime}\left(M\right)$ can be given exactly as in Section 3.4 Again, observe how the semantics of all terms of $\Lambda^{\mathtt{cps}}_{\oplus}$ 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 λ-calculus [de Carvalho 2018; Gardner 1994; Kfoury 2000; Neergaard and Mairson 2004] are well-studied. We adapt them to our probabilistic setting. The types reflect the underlying dynamics, which is simpler than that of CbV, since a term cannot be copied once evaluated. Like in the case of $\Lambda^{\mathtt{cbv}}_{\oplus}$, the type system is based on *three*, rather than *two* layers, namely arrows, intersection types, 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 $q.\mathtt{a}$ where $\mathtt{a}$ 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:

$$\begin{array}{rll} \mathtt{A}, \mathtt{B} ::= * \mid \mathcal{A} \to \mathtt{a} & & \textbf{Arrow Types} \\ \mathcal{A}, \mathcal{B} ::= [q_1.\mathtt{a}_1, \ldots, q_n.\mathtt{a}_n] n \geq 0 & & \textbf{Intersection Types} \\ \mathtt{a}, \mathtt{b} ::= \langle p_1 \mathtt{A}_1, \ldots, p_n \mathtt{A}_n \rangle, n \geq 0 & & \textbf{Type Distributions} \end{array}$$

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 judgments of the shape $\Gamma \overset{w}{\vdash} M : \mathbf{T}$, where Γ is a type context, M a term, $w \in \mathbb{Q}$ is a counter, and $\mathbf{T}$ is either $\mathtt{a}$ or $\mathcal{A}$. The notation $q.\Gamma$ is as in Section 4.1, taking into account that now if $\mathcal{A} = [q_i.\mathtt{a}_i]_{i\in I}$, $u.\mathcal{A}$ is $[(uq_i).\mathtt{a}_i]_{i\in I}$. The notion of

a tight type needs to be appropriately adapted.

Definition 28 (Tight Types and Derivations)**.** A *type* $\mathtt{a}$ is said to be *tight* if it is a multidistribution on the arrow type $*$. A derivation $\Pi \triangleright \overset{w}{\vdash} M : \mathtt{a}$ is tight whenever $\mathtt{a}$ is tight.

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 $\overset{0}{\vdash} V : \langle * \rangle$. Moreover, a degenerate form of the rule ! allows us to derive the following *for any term* M:

$$\frac{}{\overset{0}{\vdash} M : []}$$

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

$$\frac{\Gamma \overset{w}{\vdash} M : \mathbf{0}}{\Gamma \overset{w}{\vdash} MN : \mathbf{0}}$$

8.1.2 Characterising CbN Probabilistic Termination. The just introduced type system allows us to transfer all results from Section 6 to $\Lambda_{\oplus}^{\mathtt{cbn}}$. 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 $\Lambda_{\oplus}^{\mathtt{cbn}}$. 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 type 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 types. 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 $\mathtt{let}\ x = N\ \mathtt{in}\ M$. Since the argument N is typed with a multidistribution $\mathtt{c} = \langle p_k \mathcal{A}_k \rangle_{k \in K}$, the continuation M must be able to receive any $\mathcal{A}_k$. Indeed, the typing rule $\mathtt{let}$ asks for type derivations having conclusion $x : \mathcal{A}_k \vdash M : \mathtt{b}_k$ 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 $\mathcal{A}_h$ and $\mathcal{A}_l$ in $\mathtt{c}$ is simply not possible. Going to distributions, thus allowing for such a collapse, would not be a problem for *soundness*, but we would loose 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* $P \to \left\langle \frac{1}{2}P_1, \frac{1}{2}P_2 \right\rangle$. *The claim of weighted subject expansion is that, given derivations* $\Pi_i \triangleright \overset{w_i}{\vdash} P_i : \mathtt{b}_i$ *for each* i, *we can obtain a derivation* $\Pi \triangleright \overset{w_i}{\vdash} P : \mathtt{b}$, *where* $w \geq 1 + \sum \frac{1}{2} w_i$ *and* $\mathtt{b} = \frac{1}{2}\mathtt{b}_1 \uplus \frac{1}{2}\mathtt{b}_2$. *Weighted subject expansion is proved by induction on the structure of the reduction* $\to$. *The key point is the* $\mathtt{letC}$ *rule. Let us focus on it. Consider* $P := (\mathtt{let}\ x = N_1 \oplus N_2\ \mathtt{in}\ M)$ *and so* $P_i := (\mathtt{let}\ x = N_i\ \mathtt{in}\ M)$,

and consider the following type derivations for P_1 and P_2.

$$\dfrac{\overset{v_1}{\vdash} N_1 : \langle \mathcal{A} \rangle \quad \Pi_1 \triangleright x : \mathcal{A} \overset{u_1}{\vdash} M : \mathtt{b}_1}{\overset{w_1}{\vdash} \mathtt{let}\ x = N_1\ \mathtt{in}\ M : \mathtt{b}_1}\ \mathtt{let} \qquad \dfrac{\overset{v_2}{\vdash} N_2 : \left\langle \frac{1}{2}\mathcal{A} \right\rangle \quad \Pi_2 \triangleright x : \mathcal{A} \overset{u_2}{\vdash} M : \mathtt{b}_2}{\overset{w_2}{\vdash} \mathtt{let}\ x = N_2\ \mathtt{in}\ M : \frac{1}{2}\mathtt{b}_2}\ \mathtt{let}$$

By definition, $P \to \left\langle \frac{1}{2}P_1, \frac{1}{2}P_2 \right\rangle$ is derived as follow:

$$\dfrac{N \to \left\langle \frac{1}{2}N_1, \frac{1}{2}N_2 \right\rangle}{(\mathtt{let}\ x = N_1 \oplus N_2\ \mathtt{in}\ M) \to \left\langle \frac{1}{2}(\mathtt{let}\ x = N_1\ \mathtt{in}\ M), \frac{1}{2}(\mathtt{let}\ x = N_2\ \mathtt{in}\ M) \right\rangle}\ \mathtt{let}C$$

and we would like to derive a type derivation for P out of all this. By i.h.*, since $N \to \left\langle \frac{1}{2}N_i \right\rangle_{i\in I}$, we can assume that there exists a derivation $\Phi \triangleright \overset{v}{\vdash} N : \mathtt{c}$ such that $v \geq 1 + \sum \frac{1}{2}v_i$ and $\mathtt{c} = \frac{1}{2}\langle \mathcal{A} \rangle \uplus \frac{1}{2}\left\langle \frac{1}{2}\mathcal{A} \right\rangle = \left\langle \frac{1}{2}\mathcal{A}, \frac{1}{4}\mathcal{A} \right\rangle$. And indeed, by collecting Π_1 and Π_2, we have a derivation which satisfies the claim*

$$\dfrac{\overset{v}{\vdash} N : \left\langle \frac{1}{2}\mathcal{A}, \frac{1}{4}\mathcal{A} \right\rangle \quad x : (\mathcal{A} \overset{u_i}{\vdash} M : \mathtt{b}_i)_{i\in\{1,2\}}}{\overset{w}{\vdash} \mathtt{let}\ x = N\ \mathtt{in}\ M : \frac{1}{2}\mathtt{b}_1 \uplus \frac{1}{4}\mathtt{b}_2}\ \mathtt{let}$$

This is possible precisely because—due to the adoption of multidistributions*—the two occurrences of $\mathcal{A}$ are kept separated: notice that $\mathtt{b}_1$ and $\mathtt{b}_2$ 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 $\Phi \triangleright \overset{v}{\vdash} N : \mathtt{c}$ where $\mathtt{c} = \frac{1}{2}\langle \mathcal{A} \rangle + \frac{1}{2}\left\langle \frac{1}{2}\mathcal{A} \right\rangle = \left\langle \frac{3}{4}\mathcal{A} \right\rangle$. We would like to build the following derivation:*

$$\dfrac{\overset{v}{\vdash} N : \left\langle \frac{3}{4}\mathcal{A} \right\rangle \qquad \Pi}{\overset{w}{\vdash} \mathtt{let}\ x = N\ \mathtt{in}\ M : \frac{1}{2}\mathtt{b}_1 \uplus \frac{1}{4}\mathtt{b}_2}\ \mathtt{let}$$

How could we build Π, however? There is no way to merge *the two derivations Π_1 and Π_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 [Hughes et al. 1996], and linear dependent types [Dal Lago and Gaboardi 2011] have been generalised to probabilistic programming languages, and have been proved to be sound methodologies for checking almost-sure termination [Dal Lago and Grellois 2019] and positive almost-sure termination [Avanzini et al. 2019] in an higher-order setting. None of such systems is complete, however. Recently, Breuvart and Dal Lago [Breuvart and Dal Lago 2018] 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 [Coppo and Dezani-Ciancaglini 1978, 1980], and developed in a series of papers in which various notions of termination for the

λ-calculus were characterised, and the relationship with denotational semantics was thoroughly investigated [Barendregt et al. 1983; Coppo et al. 1980, 1987; Pottinger 1980]. They have also been extended to calculi besides the λ-calculus, like $\lambda\mu$-calculi [van Bakel et al. 2012] or object calculi [de'Liguoro 2001]. Besides the already discussed work by Breuvart and Dal Lago [Breuvart and Dal Lago 2018], one should also mention the work by de' Liguoro and colleagues [de'Liguoro and Piperno 1995; Dezani-Ciancaglini et al. 1993] about filter models and intersection type assignment systems for extensions of λ-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 [Gardner 1994], studied in connection with expansion variables by Carlier et al. [Carlier et al. 2004], and further analysed in their relation to normalisation by Mairson and Möller-Neergard [Neergaard and Mairson 2004]. 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 [de Carvalho 2018], and further refined by Bernadet and Lengrand [Bernadet and Lengrand 2013], and later by Accattoli et al. [Accattoli et al. 2018], and [Accattoli et al. 2019], 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 λ-calculi.

Formal verification techniques for probabilistic termination and complexity analysis are plentiful, and ranges from model checking [Etessami and Yannakakis 2009; Kobayashi et al. 2019] to abstract interpretation [Monniaux 2001], to the ranking supermartingales [Chakarov and Sankaranarayanan 2013], to amortised analysis [Ngo et al. 2018] to the interpretation method from term rewriting [Avanzini et al. 2020]. 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. [Kobayashi et al. 2019], 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). The presence of higher-order functions, however, forced us to develop new tools, since types must be more informative than just, say, ranking supermartingales. Not only the *value* or the *size* of the input matter, but also how the input *behaves* turns out to be crucial, given that it can potentially be used as a function. Looking at all this from a different perspective, we can safely say that higher-order probabilistic programs could of course be verified by translating the input program into a first-order equivalent one, then applying state-of-the art techniques designed for such a setting (e.g., [Kaminski et al. 2018; McIver and Morgan 2005]). The main advantage of thinking in terms of types, however, is that the underlying verification problem can be tackled *compositionally*, so allowing for a modular analysis. In presence of higher-order functions, one has to prove something stronger than the mere underlying termination property, namely that the program at hand satisfies the property when seen in isolation, but also behaves well when fed with functional inputs, provided those functions behave well themselves. Verification techniques designed for first-order programs are not designed with all this in mind, and encoded higher-order programs would thus be harder to verify.

The operational and denotational semantics of probabilistic λ-calculi have been studied thoroughly themselves, starting from the pioneering contributions by Sahed-Djaromi [Saheb-Djahromi 1978] and Jones and Plotkin [Jones and Plotkin 1989]. Noticeably, Ehrhard et al.'s probabilistic coherent spaces [Ehrhard et al. 2014] 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 probabilistic λ-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 [Kfoury and Wells 1999; Kobayashi and Ong 2009; Tsukada and Kobayashi 2012]. As a consequence, it would be very interesting, e.g., to study which fragments of $\Lambda_{\oplus}^{\mathtt{cbv}}$ and $\Lambda_{\oplus}^{\mathtt{cbn}}$ are expressive enough to capture recursive Markov chains [Etessami and Yannakakis 2009], 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 *multi*distributions 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 λ-calculi is outside the scope of this paper, but it is certainly something the authors would like to pursue in the foreseeable future.

ACKNOWLEDGMENTS

This work was partially supported by ANR PRC project PPS (ANR-19-CE48-0014), by ERC Consolidator Grant DIAPASoN (818616), and by MIUR PRIN ASPRA (201784YSZ5).

REFERENCES

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

Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. 2019. Types by Need. In *Proc. of ESOP 2019 (LNCS)*, Vol. 11423. 410–439. https://doi.org/10.1007/978-3-030-17184-1_15

Martin Avanzini, Ugo Dal Lago, and Alexis Ghyselen. 2019. Type-Based Complexity Analysis of Probabilistic Functional Programs. In *Proc. of LICS 2019*. 1–13. https://doi.org/10.1109/LICS.2019.8785725

Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. 2020. On probabilistic term rewriting. *Sci. Comput. Program.* 185 (2020). https://doi.org/10.1016/j.scico.2019.102338

Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. 1983. A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic* 48, 4 (1983), 931–940. https://doi.org/10.2307/2273659

Alexis Bernadet and Stéphane Lengrand. 2013. Non-idempotent intersection types and strong normalisation. *Log. Methods Comput. Sci.* 9, 4 (2013). https://doi.org/10.2168/LMCS-9(4:3)2013

Patrick Billingsley. 1979. *Probability and measure.* John Wiley and Sons, New York.

Ales Bizjak and Lars Birkedal. 2015. Step-Indexed Logical Relations for Probability. In *Proc. of FoSSaCS*. 279–294. https://doi.org/10.1007/978-3-662-46678-0_18

Olivier Bournez and Florent Garnier. 2006. Proving Positive Almost Sure Termination Under Strategies. In *Rewriting Techniques and Applications, RTA*. 357–371. https://doi.org/10.1007/11805618_27

Pierre Brémaud. 2017. *Discrete Probability Models and Methods.* Springer. https://doi.org/10.1007/978-3-319-43476-6

Flavien Breuvart and Ugo Dal Lago. 2018. On Intersection Types and Probabilistic Lambda Calculi. In *Proc. of PPDP 2018*. 8:1–8:13. https://doi.org/10.1145/3236950.3236968

Sébastien Carlier, Jeff Polakow, J. B. Wells, and A. J. Kfoury. 2004. System E: Expansion Variables for Flexible Typing with Linear and Non-linear Types and Intersection Types. In *Proc. of ESOP 2004 (LNCS)*, Vol. 2986. Springer, 294–309. https://doi.org/10.1007/978-3-540-24725-8_21

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Proc. of CAV 2013 (LNCS)*, Vol. 8044. Springer, 511–526. https://doi.org/10.1007/978-3-643-39799-8_34

Mario Coppo and Mariangiola Dezani-Ciancaglini. 1978. A new type assignment for lambda-terms. *Archiv für mathematische Logik und Grundlagenforschung* 19, 1 (1978), 139–156. https://doi.org/10.1007/BF02011875

Mario Coppo and Mariangiola Dezani-Ciancaglini. 1980. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic* 21, 4 (10 1980), 685–693. https://doi.org/10.1305/ndjfl/1093883253

Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. 1980. Principal type schemes and lambda-calculus semantics. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 535–560.

Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. 1981. Functional Characters of Solvable Terms. *Math. Log. Q.* 27, 2-6 (1981), 45–58. https://doi.org/10.1002/malq.19810270205

Mario Coppo, Mariangiola Dezani-Ciancaglini, and Maddalena Zacchi. 1987. Type Theories, Normal Forms and D_∞-Lambda-Models. *Inf. Comput.* 72, 2 (1987), 85–116. https://doi.org/10.1016/0890-5401(87)90042-3

Patrick Cousot. 1997. Types as Abstract Interpretations. In *Proc. of POPL 1997*. 316–331. https://doi.org/10.1145/263699.263744

Ugo Dal Lago and Beniamino Accattoli. 2017. Encoding Turing Machines into the Deterministic Lambda-Calculus. *CoRR* abs/1711.10078 (2017). http://arxiv.org/abs/1711.10078

Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. 2020. Intersection Types and (Positive) Almost-Sure Termination (Extended version). *CoRR* 2010.12689 (2020). http://arxiv.org/abs/2010.12689

Ugo Dal Lago and Marco Gaboardi. 2011. Linear Dependent Types and Relative Completeness. *Log. Methods Comput. Sci.* 8, 4 (2011). https://doi.org/10.2168/LMCS-8(4:11)2012

Ugo Dal Lago and Charles Grellois. 2019. Probabilistic Termination by Monadic Affine Sized Typing. *ACM Trans. Program. Lang. Syst.* 41, 2 (2019), 10:1–10:65. https://doi.org/10.1145/3293605

Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. 2014. On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. of POPL 2014*. 297–308. https://doi.org/10.1145/2535838.2535872

Ugo Dal Lago and Margherita Zorzi. 2012. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.* 46, 3 (2012), 413–450. https://doi.org/10.1051/ita/2012012

Daniel de Carvalho. 2018. Execution time of λ-terms via denotational semantics and intersection types. *Math. Struct. Comput. Sci.* 28, 7 (2018), 1169–1203. https://doi.org/10.1017/S0960129516000396 Availabel in preprint form from 2009 https://arxiv.org/abs/0905.4251.

Karel De Leeuw, Edward F Moore, Claude E Shannon, and Norman Shapiro. 1956. Computability by probabilistic machines. *Automata studies* 34 (1956), 183–198.

Ugo de'Liguoro. 2001. Characterizing Convergent Terms in Object Calculi via Intersection Types. In *Proc. of TLCA 2001 (LNCS)*, Vol. 2044. Springer, 315–328. https://doi.org/10.1007/3-540-45413-6_25

Ugo de'Liguoro and Adolfo Piperno. 1995. Non Deterministic Extensions of Untyped Lambda-Calculus. *Inf. Comput.* 122, 2 (1995), 149–177. https://doi.org/10.1006/inco.1995.1145

Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Adolfo Piperno. 1993. Filter Models for a Parallel and Non Deterministic Lambda-Calculus. In *Proc. of MFCS 1993*. 403–412. https://doi.org/10.1007/3-540-57182-5_32

Thomas Ehrhard, Christine Tasson, and Michele Pagani. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Proc. of POPL 2014*. ACM, 309–320. https://doi.org/10.1145/2535838.2535865

Kousha Etessami and Mihalis Yannakakis. 2009. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM* 56, 1 (2009), 1:1–1:66. https://doi.org/10.1145/1462153.1462154

Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Proc. of POPL 2015*. 489–501. https://doi.org/10.1145/2676726.2677001

Philippa Gardner. 1994. Discovering Needed Reductions Using Type Theory. In *Proc. of TACS '94, (LNCS)*, Vol. 789. Springer, 555–574. https://doi.org/10.1007/3-540-57887-0_155

John Gill. 1977. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.* 6, 4 (1977), 675–695. https://doi.org/10.1137/0206049

Jean-Yves Girard. 1971. 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*. Studies in Logic and the Foundations of Mathematics, Vol. 63. Elsevier, 63 – 92. https://doi.org/10.1016/S0049-237X(08)70843-7

Shafi Goldwasser and Silvio Micali. 1984. Probabilistic encryption. *Journal of computer and system sciences* 28, 2 (1984), 270–299. https://doi.org/10.1016/0022-0000(84)90070-9

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI*. 220–229.

Jean Goubault-Larrecq. 2015. Full Abstraction for Non-Deterministic and Probabilistic Extensions of PCF I: the Angelic Cases. *Journal of Logic and Algebraic Methods in Programming* 84 (2015), 155–184. https://doi.org/10.1016/j.jlamp.2014.09.003

John Hughes, Lars Pareto, and Amr Sabry. 1996. Proving the Correctness of Reactive Systems Using Sized Types. In *Proc. of POPL 1996*. ACM Press, 410–423. https://doi.org/10.1145/237721.240882

Claire Jones and Gordon D. Plotkin. 1989. A Probabilistic Powerdomain of Evaluations. In *Proc. of LICS 1989*. 186–195. https://doi.org/10.1109/LICS.1989.39173

Achim Jung and Regina Tix. 1998. The troublesome probabilistic powerdomain. *Electr. Notes Theor. Comput. Sci.* 13 (1998), 70–91. https://doi.org/10.1016/S1571-0661(05)80216-6

Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2019. On the hardness of analyzing probabilistic programs. *Acta Informatica* 56, 3 (2019), 255–285. https://doi.org/10.1007/s00236-018-0321-1

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5 (2018), 30:1–30:68. https://doi.org/10.1145/3208102

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

A. J. Kfoury and J. B. Wells. 1999. Principality and Decidable Type Inference for Finite-Rank Intersection Types. In *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, Andrew W. Appel and Alex Aiken (Eds.). ACM, 161–174. https://doi.org/10.1145/292540.292556

Naoki Kobayashi. 2009. 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*. 416–428. https://doi.org/10.1145/1480881.1480933

Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. 2019. On the Termination Problem for Probabilistic Higher-Order Recursive Programs. In *Proc. of LICS 2019*. 1–14. https://doi.org/10.1109/LICS.2019.8785679

Naoki Kobayashi and C.-H. Luke Ong. 2009. A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes. In *Proc. of LICS 2009*. 179–188. https://doi.org/10.1109/LICS.2009.29

Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.* 22, 3 (1981), 328–350. https://doi.org/10.1016/0022-0000(81)90036-2

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer. https://doi.org/10.1007/b138392

David Monniaux. 2001. An Abstract Analysis of the Probabilistic Termination of Programs. In *Proc. of SAS 2001*. 111–126. https://doi.org/10.1145/360204.360211

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press. https://doi.org/10.1017/cbo9780511814075

Peter Møller Neergaard and Harry G. Mairson. 2004. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In *Proc. of ICFP 2004*. 138–149. https://doi.org/10.1145/1016850.1016871

Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: resource analysis for probabilistic programs. In *Proc. of PLDI 2018*. 496–512. https://doi.org/10.1145/3192366.3192394

Piergiorgio Odifreddi. 1989. *Classical Recursion Theory*. Elsevier.

C.-H. Luke Ong. 2006. On Model-Checking Trees Generated by Higher-Order Recursion Schemes. In *Proc. of LICS 2006*. 81–90. https://doi.org/10.1109/LICS.2006.38

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

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

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

Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA. https://doi.org/10.1002/9780470316887

Michael O Rabin. 1963. Probabilistic automata. *Information and control* 6, 3 (1963), 230–245. https://doi.org/10.1016/S0019-9958(63)90290-0

N. Saheb-Djahromi. 1978. Probabilistic LCF. In *Proc. of MFCS 1978 (LNCS)*, Vol. 64. 442–451. https://doi.org/10.1007/3-540-08921-7_92

Eugene S. Santos. 1969. Probabilistic Turing machines and computability. *Proc. Amer. Math. Soc.* 22, 3 (1969), 704–710.

Morten Heine Sørensen and Pawel Urzyczyn. 1989. *Lectures on the Curry-Howard Isomorphism*. Elsevier. https://doi.org/10.1016/S0049-237X(06)80005-4

David Tolpin, Jan-Willem van de Meent, and Frank D. Wood. 2015. Probabilistic Programming in Anglican. In *Proc. of ECML PKDD 2015 (LNCS)*, Vol. 9286. Springer, 308–311. https://doi.org/10.1007/978-3-319-23461-8_36

Takeshi Tsukada and Naoki Kobayashi. 2012. An Intersection Type System for Deterministic Pushdown Automata. In *Proc. of TCS 2012*. 357–371. https://doi.org/10.1007/978-3-642-33475-7_25

Steffen van Bakel, Franco Barbanera, and Ugo de'Liguoro. 2012. Characterisation of Strongly Normalising lambda-mu-Terms. In *Proc. of , ITRS 2012 (EPTCS)*, Vol. 121. 1–17. https://doi.org/10.4204/EPTCS.121.1

Christopher Wadsworth. 1980. Some unusual λ-calculus numeral systems. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J.P. Seldin and J.R. Hindley (Eds.). Academic Press.