On Linear Dependent Types and Probabilistic Termination

Ugo Dal Lago[1] and Alexis Ghyselen[2]

1 Università di Bologna, Italy & INRIA Sophia Antipolis, France
ugo.dallago@unibo.it
2 ENS Paris-Saclay, France
ghyselen.alexis@gmail.com

Abstract

Linear dependent types have been shown to be a very powerful methodology for inferring the time complexity of deterministic functional programs. We present some preliminary results on the application of linear dependency to *probabilistic* functional programs. We will in particular give some hints at how linear dependent types offer a leverage towards guaranteeing almost sure termination.

1998 ACM Subject Classification Dummy classification – please refer to `http://www.acm.org/about/class/ccs98-html`

Keywords and phrases λ-calculus, probabilistic termination, linear dependent types

Digital Object Identifier 10.4230/LIPIcs.DICE.2018.23

1 Introduction

Interactions between computer science and probability theory are very pervasive and fruitful. Probability theory offers models that enable system abstraction, but also suggests a new model *of computation*, like in randomized computation or cryptography [5]. All this has stimulated the study of probabilistic programming languages: probabilistic imperative languages and λ-calculi [10], have indeed been introduced and studied for at least forty years. Among the many ways probabilistic choice can be modeled in programming, the simplest one consists in endowing the language of programs with an operator modeling sampling from (one or many) distributions. Fair, binary, probabilistic choice is for example perfectly sufficient to get universality if the underlying programming language is itself universal [11, 4]. This is precisely what happened in the realm of the λ-calculus [10, 7].

Termination is a crucial property of programs, and remains desirable in a probabilistic setting, e.g. in probabilistic programming [6] where inference algorithms often rely on the underlying program to terminate. However, one needs first of all to understand *what it means* for a probabilistic computation to terminate, i.e., how termination should be defined. If one wants to stick to a *qualitative* definition, almost-sure termination is a well-known answer: a probabilistic computation is said to almost-surely terminate iff divergence, although possible, has null probability. One could even go beyond and require *positive* almost-sure termination, which asks the average time to termination to be finite. This is well-known to be stronger than almost sure termination. Recursion-theoretically, checking (positive) almost-sure termination is harder than checking termination in deterministic computation, where termination is at least recursively enumerable, although undecidable: in a universal probabilistic imperative programming language, almost sure termination is Π^0_2 complete, while positive almost-sure termination is Σ^0_2 complete [8].

© Ugo Dal Lago and Alexis Ghyselen;
licensed under Creative Commons License CC-BY

A few works analyze how termination analysis can be carried out automatically, despite its recursion-theoretic hardness. The only attempt to do that via types is due to the first author and Grellois [9], and is based on size types. In that setting, monadic affine sized types are shown sound for almost sure termination. In a purely deterministic setting, on the other hand, linear dependent types are well-known to be quite powerful, to the point of being *relatively complete* as a methodology for complexity analysis [2].

In this paper, we study how linear dependent types can be turned into a sound methodology for almost sure termination. The object language is an affine variation on Plotkin's PCF enriched with an operator for binary probabilistic choice. The language and its operational semantics are described in the following section.

2 A Simple Probabilistic Functional Programming Language

We consider an affine λ-calculus with constructors for integers, a fixpoint and a binary probabilistic choice operator. For the sake of simplicity, we present here an ordinary fixpoint operator, but we can extend this work to an operator for simultaneous fixpoints, to increase expressivity. We adopt a call-by-value evaluation, and so we distinguish terms and values.

▶ **Definition 1** (Terms and Values)**.** Terms and values are defined as follows :

$$\begin{aligned} t,u,s &::= x \;\big|\; v \;\big|\; t\ u \;\big|\; \mathtt{succ}(t) \;\big|\; \mathtt{pred}(t) \;\big|\; \mathtt{ifz}\ s\ \mathtt{then}\ u\ \mathtt{else}\ t \;\big|\; t \oplus_p u \\ v &::= \underline{n} \;\big|\; \lambda x.t \;\big|\; \mathtt{fix}\ x.t \end{aligned}$$

with $p \in \mathbb{Q} \cap [0,1]$. An integer $n \in \mathbb{N}$ is represented in this calculus by the constant $\underline{n}$. Terms which are not values are said to be *active*.

For example, the term describing the random walk that stops when it reaches 0, with a probability $\frac{2}{3}$ to decrease at each step is $fix f.\lambda x.\mathtt{ifz}\ x\ \mathtt{then}\ \underline{0}\ \mathtt{else}\ f\ \mathtt{pred}(x) \oplus_{\frac{2}{3}} f\ \mathtt{succ}(x)$

▶ **Definition 2** (Types)**.** We consider usual simple types for the λ-calculus with a base type Nat for integers: $T,U ::= \mathsf{Nat} \;\big|\; T \multimap U$

We impose affinity in the calculus for higher-order types. This means that all variables with a type $T \multimap U$ appears at most once in any probabilistic branch. Defining the operational semantics of the just introduced language requires introducing the notion of distribution.

▶ **Definition 3** (Distributions)**.** A *distribution* $\mathcal{D}$ on a countable set X is a function $X \to [0,1]$ such that $\sum \mathcal{D} = \sum_{x\in X} \mathcal{D}(x) \leq 1$. A distribution is *finite* when its *support* $\{x \mid \mathcal{D}(x) > 0\}$ is finite. We denote a distribution $\mathcal{D}$ by $\{(x)^{\mathcal{D}(x)} \mid x \in X\}$. With this notation, we often omit some elements with probability 0. For a distribution $\mathcal{D}$ and $0 \leq p \leq 1$, we write $p \cdot \mathcal{D}$ for the distribution $\{x^{p\cdot\mathcal{D}(x)} \mid x \in X\}$. For two distributions $\mathcal{D}, \mathcal{E}$ on X, we note $\mathcal{D} + \mathcal{E}$ the function $\{(x)^{\mathcal{D}(x)+\mathcal{E}(x)} \mid x \in X\}$. Then, we note $\mathcal{D} \oplus_p \mathcal{E}$ the distribution $p \cdot \mathcal{D} + (1-p) \cdot \mathcal{E}$. Finally, we say that $\mathcal{D} \leq \mathcal{E}$ if $\forall x \in X, \mathcal{D}(x) \leq \mathcal{E}(x)$.

Any term distribution can be split into two components, the first having a support made of values, the second a support made of active terms.

▶ **Definition 4** (Value Decomposition)**.** We define the *value decomposition* of a distribution on terms $\mathcal{D}$ as the pair $(\mathcal{D}_v, \mathcal{D}_t)$ where $\mathcal{D}_v$ is the restriction of $\mathcal{D}$ to values, while $\mathcal{D}_t$ is the restriction of the function $\mathcal{D}$ to active terms. We write $\mathcal{D} =_{vd} \mathcal{D}_v + \mathcal{D}_t$ to denote that $(\mathcal{D}_v, \mathcal{D}_t)$ is the value decomposition of $\mathcal{D}$.

$$\frac{}{(\lambda x.t)\ v \to \{(t[x := v])^1\}} \qquad \frac{}{\mathtt{succ}(\underline{n}) \to \{(\underline{n+1})^1\}}$$

$$\frac{}{\mathtt{pred}(\underline{n}) \to \{(\underline{pred(n)})^1\}} \qquad \frac{}{\mathtt{ifz}\ \underline{0}\ \mathtt{then}\ t\ \mathtt{else}\ u \to \{t^1\}}$$

$$\frac{}{\mathtt{ifz}\ \underline{n+1}\ \mathtt{then}\ t\ \mathtt{else}\ u \to \{u^1\}} \qquad \frac{}{t \oplus_p u \to \{t^p, u^{1-p}\}}$$

$$\frac{}{(\mathtt{fix}\ x.t)\ v \to \{(t[x := \mathtt{fix}\ x.t]\ v)^1\}} \qquad \frac{t \to \{t_i^{p_i} \mid i \in \mathcal{I}\}}{t\ u \to \{(t_i\ u)^{p_i} \mid i \in \mathcal{I}\}}$$

$$\frac{u \to \{u_i^{p_i} \mid i \in \mathcal{I}\}}{v\ u \to \{(v\ u_i)^{p_i} \mid i \in \mathcal{I}\}} \qquad \frac{t \to \{t_i^{p_i} \mid i \in \mathcal{I}\}}{\mathtt{succ}(t) \to \{\mathtt{succ}(t_i)^{p_i} \mid i \in \mathcal{I}\}}$$

$$\frac{s \to \{s_i^{p_i} \mid i \in \mathcal{I}\}}{\mathtt{ifz}\ s\ \mathtt{then}\ t\ \mathtt{else}\ u \to \{(\mathtt{ifz}\ s_i\ \mathtt{then}\ t\ \mathtt{else}\ u)^{p_i} \mid i \in \mathcal{I}\}}$$

$$\frac{t \to \{t_i^{p_i} \mid i \in \mathcal{I}\}}{\mathtt{pred}(t) \to \{\mathtt{pred}(t_i)^{p_i} \mid i \in \mathcal{I}\}} \qquad \frac{\mathcal{D} =_{vd} \mathcal{D}_v + \{t_i^{p_i} \mid i \in \mathcal{I}\} \qquad \forall i \in \mathcal{I}, t_i \to \mathcal{E}_i}{\mathcal{D} \to \mathcal{D}_v + \sum_{i \in \mathcal{I}} p_i \cdot \mathcal{E}_i}$$

Figure 1 Reductions Rules on Distributions

We are now in a position to define a reduction relation $\to$ between distributions. The rules are described in Figure 1. Term substitution $t[x := v]$ is defined is the usual way. This relation is first defined for terms and then lifted to distributions with the last rule. We note $\to^*$ the reflexive and transitive closure of $\to$. We also need another relation $\Rightarrow_V$. If $\mathcal{D} \to^* \mathcal{E}$ and $\mathcal{E} =_{vd} (\mathcal{E}_v + \mathcal{E}_t)$, then we have $\mathcal{D} \Rightarrow_V \mathcal{E}_v$. The rules give us that if $(\mathcal{E} =_{vd} \mathcal{E}_v + \mathcal{E}_t) \to^* (\mathcal{F} =_{vd} \mathcal{F}_v + \mathcal{F}_t)$ then $\mathcal{E}_v \leq \mathcal{F}_v$. We can then define the semantics:

▶ **Definition 5** (Semantics of a Term). The *semantics* of a term is a distribution on values defined as $[\![t]\!] = sup\{\mathcal{D} \mid t \Rightarrow_V \mathcal{D}\}$. This is a well-posed definition because distributions form an ω**CPO**.

3 Linear Dependency: Some Typing Rules

We define a linear dependent type system in order to control the reduction procedure in our calculus. Linear dependent types were first introduced by Gaboardi and the first author [2] and later adapted to call-by-value PCF [3]. The type system can be seen as a variation of affine sized types for almost sure termination introduced by Grellois and the first author [9].

▶ **Definition 6** (Linear Dependent Types). Linear dependent types σ, τ are defined by :

$$\sigma, \tau ::= A \;\big|\; \mathsf{Nat}[I, J] \qquad A, B ::= \sigma \multimap \tau \qquad I ::= a \;\big|\; f(I_1, \ldots, I_n)$$

An expression I is called an *index*. The notations f in indexes denote functions $f : \mathbb{N}^n \to \mathbb{N}$. They are defined by a fixed set of rewriting rules and include the addition and the subtraction. Informally, given an instantiation of all index variables by integers, an index denotes an integer, and the type $\mathsf{Nat}[I, J]$ denotes the set of all integers $\underline{n}$ such that $I \leq n \leq J$. The substitution of the occurrences of a in I by J, denoted $I\{J/a\}$, is defined in the usual way.

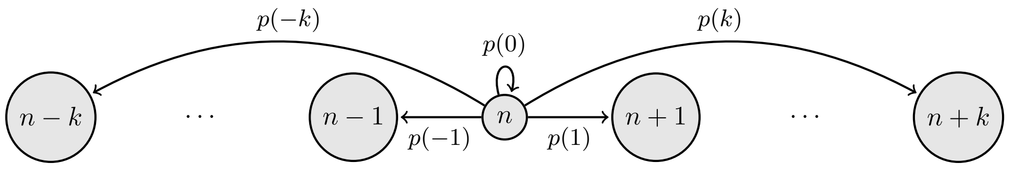
A typing judgment has the form $\phi; \Phi; \Gamma \mid \Theta \vdash t : \sigma$. ϕ is the set of all index variables free in Φ, Γ, Θ and σ. Φ is a set of constraints on indexes. This is used for example in the typing rule for `ifz`, in which choosing a branch restraints the possible value of the integer, and thus imposes a constraint on indexes. Γ is a sequence of hypotheses $y_1 : \sigma_1, \ldots, y_m : \sigma_m$. And Θ is a *distribution context*, that we will describe briefly for the fixpoint rule.

Let us sketch the most important rule for the typing system, the fixpoint rule.

$$\frac{p : \{-k \dots, 0, \dots, k\} \to [0,1] \text{ is a probability distribution} \qquad (b,\phi);\Phi;\ell\Gamma \mid x : \{1 \leq b \wedge b+l \geq 0, B\{b+l/b\})^{p(l)} \mid -k \leq l \leq k\} \vdash t : B}{\phi;\Phi;\ell\Gamma \mid \cdot \vdash \texttt{fix } x.t : B\{I/b\}}$$

In this rule, $\ell\Gamma$ is a sequence of hypotheses $y_1 : \mathsf{Nat}[I_1, J_1], \dots, y_m : \mathsf{Nat}[I_m, J_m]$. b denotes a fresh index variable that is used in order to describe the recursive calls. For example, in the random walk defined previously, b would represent the current state of the random walk. Then, we instantiate this index variable b by an actual index I. x is given a type of a special kind: a distribution of the form $\{(C_l, B_l)^{p(l)} \mid -k \leq l \leq k\}$. Informally, that means that x has probability $p(l)$ to have type B_l if you can prove the condition C_l. In the typing rule, this condition expresses that when $b = 0$ the computation must terminate and that one cannot decrease b below 0. This particular kind of context is also important for the probabilistic choice, in which we can separate such a distribution Θ into two distributions Ψ_1 and Ψ_2 such that $\Theta = \Psi_1 \oplus_p \Psi_2$.

To any instance of this typing rule, we associate a particular probabilistic process. Informally, this process describes the values taken by b throughout a computation. For the fixpoint expressed above, we construct an infinite Markov chain on integers, with for all integers $n \neq 0$ those outgoing transitions, with the convention that we identify nodes with negative values and the zero node :



For a fixpoint with one variable, this process is a random walk on $\mathbb{N}$ with transitions modifying the integer by a bounded value. For the simultaneous fixpoint, we work with discrete-time quasi-birth-deaths processes (QBD), or equivalently, probabilistic one-counter automata. QBDs are a generalization of random walks on $\mathbb{N}$ to Markov chains on $Q \times \mathbb{N}$. Q is a finite set of states representing here the different variables in the simultaneous fixpoint, and the integer is used as an unbounded counter. We say that a QBD of this kind *almost surely terminates* when the probability of reaching the value 0 for the counter is 1, independently of the initial state. Please notice that checking whether a QBD almost surely terminates is known to be decidable. In fact, it can be checked in polynomial time [1].

4 Almost Sure Termination

We show in this section how to get to almost sure termination in this calculus by modifying the fixpoint rule defined previously. First, let us define almost sure termination.

▶ **Definition 7** (Almost Sure Termination (AST)). We say that a term t is *almost surely terminating* when $\sum [\![t]\!] = 1$.

With the typing rule for fixpoint defined previously, we can find typable terms that are not AST, such as $\texttt{fix } x.x$. In a non-probabilistic setting, termination of the calculus could be ensured by restricting fixpoints to terminating fixpoints. In the same way, what matters if one wants our system to guarantee AST is that fixpoints are AST. Consequently, we modify the fixpoint rule: we impose that the random walk, or more generally the QBD, associated to the fixpoint rule is itself AST. We can then prove that the typing system satisfies an essential propriety for the λ-calculus, subject reduction.

▶ **Theorem 8** (Subject Reduction). *If $\phi; \Phi; \Gamma \mid \Theta \vdash t : \sigma$ and $t \to \{t_i^{p_i} \mid i \in \mathcal{I}\}$ then $\forall i \in \mathcal{I}$, there exists Θ_i such that $\phi; \Phi; \Gamma \mid \Theta_i \vdash t_i : \sigma$ and $\Theta = \sum_{i\in\mathcal{I}} p_i \cdot \Theta_i$.*

Then, using a reducibility argument greatly inspired from [9], we can prove that almost sure termination holds for typed terms.

▶ **Theorem 9** (Almost Sure Termination). *Suppose that $\phi; \Phi; \cdot \mid \cdot \vdash t : \sigma$, if Φ is satisfiable then t is AST.*

5 Conclusion

We are currently exploring further questions about linear dependent types and probabilistic termination: we will report about that at the workshop in case we reach a definite conclusion about those.

First, we are looking at a form of *completeness* of our calculus with respect to discrete-time QBDs, provided an operator for simultaneous fixpoints is available. In other words, it seems that any state-transforming map which can be computed by QBDs can also be computed by a term of our calculus.

Moreover, we are looking at whether one could ensure *positive* almost sure termination by way of linear dependent types, that is to say not only that the probability of divergence should be null, but that the average computation time must be finite. This property is harder to guarantee than almost sure termination, in the sense that it is not even compositional: the composition of two positive almost sure terminating functions is not necessarily positively almost surely terminating. This means that some sophisticated combinatorial arguments is indeed necessary to get the constraint that we need.

References

1 Tomás Brázdil, Václav Brožek, Kousha Etessami, Antonín Kučera, and Dominik Wojtczak. One-counter markov decision processes. In *Proc. of SODA*, pages 863–874, 2010.

2 Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *LICS*, pages 133–142, 2011.

3 Ugo Dal Lago and Barbara Petit. Linear dependent types in a call-by-value scenario. *SCP*, 84:77–100, 2014.

4 Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO: ITA.*, 46(3):413–450, 2012.

5 Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, 1984.

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

7 Claire Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *LICS*, pages 186–195, 1989.

8 Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost-sure termination. In *MFCS*, volume 9234 of *LNCS*, pages 307–318, 2015.

9 Ugo Dal Lago and Charles Grellois. Probabilistic termination by monadic affine sized typing. In *Prof. of ESOP 2017*, pages 393–419, 2017.

10 Nasser Saheb-Djahromi. Probabilistic LCF. In *MFCS*, pages 442–451, 1978.

11 Eugene S Santos. Probabilistic Turing machines and computability. *Proc. of the AMS*, 22(3):704–710, 1969.