Two Type-Theoretic Approaches to Probabilistic Termination

Ugo Dal Lago
University of Bologna
INRIA Sophia Antipolis
ugo.dallago@unibo.it

Charles Grellois
INRIA Sophia Antipolis
charles.grellois@inria.fr

This talk will be given by Charles Grellois.

Introduction. Probabilistic models are more and more pervasive in computer science [16, 18, 19]. Moreover, the concept of algorithm, originally assuming determinism, has been relaxed so as to allow probabilistic evolution since the very early days of theoretical computer science [15]. All this has given impetus to research on probabilistic programming languages, which however have been studied at a large scale only in the last twenty years, following advances in randomized computation [17], cryptographic protocol verification [2, 3], and machine learning [11]. Probabilistic programs can be seen as ordinary programs in which specific instructions are provided to make the program evolve probabilistically rather than deterministically. The typical example are instructions for sampling from a given distribution toolset, or for conditioning probabilistic choice on the values observed concretely. Our line of work studies *functional* probabilistic programming, following the recent and successful introduction of languages such as Anglican [20] and Church [11].

One of the most crucial properties a program should satisfy is *termination*: the execution process should be guaranteed to eventually end. In (non)deterministic computation, termination is a boolean predicate on programs: any (non)deterministic program either terminates – in must or may sense in the nondeterministic case – or it does not. In probabilistic programs, on the other hand, any terminating computation path is attributed a probability, and thus termination becomes a *quantitative* property. It is therefore natural to consider a program terminating when its terminating paths form a set of measure one or, equivalently, when it terminates with maximal probability. This is dubbed "almost sure termination" (AST for short) in the literature [4], and many techniques for automatically and semi-automatically checking programs for AST have been introduced in the last years [9, 10, 7, 6]. All of them, however, focus on imperative programs; while probabilistic functional programming languages are nowadays among the most successful ones in the realm of probabilistic programming [11]. It is not clear at all whether the existing techniques for imperative languages could be easily applied to functional ones, especially when higher-order functions are involved.

In this talk, we introduce two type systems for a simple *affine and probabilistic* λ-calculus with recursion which are sound for AST: typable terms are AST. Both systems are not complete, as this would make them undecidable. In future work, we will study type inference for both systems and we hope that it will be decidable at least for the first. The first type system features a monadic system of *distribution* *types*, built from the systems of *sized types* introduced in the deterministic case [13, 1], and is already presented in a long version [8] and in a ESOP 2017 paper [14]. The second type system is based on Xi's approach using dependent types [21] and is ongoing work. It is more general than the first for that it allows terms to contain conditional statements, whereas the first system only allows a form of pattern-matching on integers corresponding to a test of equality to zero. The use of dependent types allows to formulate the second system without needing to introduce distributions of types in the inference rules.

First System: Monadic Affine Sized Types. The first system is a system of monadic affine sized types which can be seen as a non-trivial variation on Hughes et al.'s sized types [13], whose main novelties are the following:

- Types are generalised so as to be *monadic*, this way encapsulating the kind of information we need to type non-trivial examples. This information, in particular, is taken advantage of when typing recursive programs.
- Typing rules are *affine*: higher-order variables cannot be freely duplicated. This is quite similar to what happens when characterising polynomial time functions by restricting higher-order languages akin to the λ-calculus [12]. Without affinity, the type system is bound to be unsound for AST.

AST is checked by a special guard on the typing rules for letrec:

$$\mathsf{letrec}\quad \frac{\begin{array}{c}\left\{\,(\mathsf{Nat}^{\mathfrak{s}_j} \to \nu[\mathfrak{s}_j/\mathfrak{i}])^{p_j} \;\mid\; j \in \mathscr{J}\,\right\} \text{induces an AST sized walk}\\ \Gamma \,|\, f : \left\{\,(\mathsf{Nat}^{\mathfrak{s}_j} \to \nu[\mathfrak{s}_j/\mathfrak{i}])^{p_j} \;\mid\; j \in \mathscr{J}\,\right\} \vdash V : \mathsf{Nat}^{\widehat{\mathfrak{i}}} \to \nu[\widehat{\mathfrak{i}}/\mathfrak{i}]\end{array}}{\Gamma, \Delta \,|\, \Theta \vdash \mathsf{letrec}\, f \;=\; V : \mathsf{Nat}^{\mathfrak{r}} \to \nu[\mathfrak{r}/\mathfrak{i}]}$$

The notion of AST sized walk defined here means that the recursive calls of the function generate a Markovian process, which should be almost-surely terminating. Let us illustrate this notion of sized walk on the example of a biased random walk

$$M_{BIAS} \;=\; \left(\mathsf{letrec}\, f \;=\; \lambda x.\mathsf{case}\; x \;\mathsf{of}\; \left\{\mathsf{S} \to \lambda y.f(y) \oplus_{\frac{2}{3}} (f(\mathsf{S}\,\mathsf{S}\,y))) \;\mid\; 0 \to 0\right\}\right)\; \underline{n} \qquad (1)$$

which calls itself recursively on an input of size lesser by 1 with probability $\frac{2}{3}$ and on size greater than 1 with probability $\frac{1}{3}$. The associated sized walk is the random process which loops forever on 0 and, on an integer $n+1$, jumps to n with probability $\frac{2}{3}$ and to $n+2$ with probability $\frac{1}{3}$. This sized walk is AST, meaning that it reaches 0 with probability 1, and this implies in turn, by soundness of our type system, that the term itself is AST. We can also type examples in which the termination argument is more involved than a strict decreasing ranking function: for instance, the term

$$M_{UNB} \;=\; \left(\mathsf{letrec}\, f \;=\; \lambda x.\mathsf{case}\; x \;\mathsf{of}\; \left\{\mathsf{S} \to \lambda y.f(y) \oplus_{\frac{1}{2}} (f(\mathsf{S}\,\mathsf{S}\,y))) \;\mid\; 0 \to 0\right\}\right)\; \underline{n} \qquad (2)$$

is AST because the associated sized walk (which is the usual one-dimensional random walk) is AST, even if the average size of the recursively-called input is not decreasing.

This first type system is already presented in a long version [8] and in a ESOP 2017 paper [14] ; we plan to briefly introduce it as a first step towards its generalization.

Second System: Using Dependent Types. The second system is a generalization of the first, based no longer on sized types, but on dependent types in the spirit of Xi's work [21]. First of all, we allow a slightly more general language than in the first case. In the previous type system, lambda-terms where manipulating natural numbers defined from successor and zero constants (the idea being that the system could be adapted to all inductive types, natural numbers being a simple example, but this generalization is left for future work), together with a pattern-matching operation. Here we consider a signature of constants which can contain integers (as built-in constants) and relations on them, and we introduce an “if” construct which is more general than the simple pattern-matching of the previous case. In addition to this, this second type system has two main novelties compared to the first one:

- Type dependency allows to get rid of distribution types. Indeed, distributions of types where needed to type precisely a probabilistic sum of terms $M \oplus_p N$ with different size informations on M and N (without this subtlety, the system was bound to capture termination and not *almost-sure* termination). The use of sum types allows to give a sum type to both M and N in this case, and thus to give this same sum type to $M \oplus_p N$. The size information still appears, but as a size annotation inside the term. This greatly simplifies the system.
- On the other hand, termination is no longer checked by a sized walk, which was a kind of slightly generalized one-dimensional random walk, but by a much more subtle machinery called *probabilistic transition systems* (PTS). These are systems in which the transitions are guarded by logical formulas, and after each guard a probabilistic choice is made on the successor states. We use these PTS to model more accurately programs and notably use as much information as possible from the choices made in the if constructs. For each letrec construct, a PTS is built from the definition of the recursive function of interest; and the typing rule for letrec checks that this PTS is AST. We rely on existing results on termination of PTS such as [5] to establish the fact that a given PTS is AST.

This second system is ongoing work. If time allows, we will explain the main changes in the reducibility proof from the previous case.

Future work. There are two main directions for future work. The first one is to extend the class of typable problems. It will certainly be difficult to get rid of the affinity constraint, as it would require to guard the letrec rule with Markovian processes which are very complex and do not seem to have been studied by specialists. However, extensions to programs with all inductive data types seem very easy, and we have ideas for extensions to programs with *coinductive* data types. It would also be possible to consider programs enriched with a non-deterministic operator in addition to the probabilistic one. The second direction would be to adapt the type systems existing for capturing complexity classes, and notably PTIME, to capture programs terminating in probabilistic polynomial time (or the probabilistic analog of other complexity classes captured by type systems).

Structure of the Talk

We plan to give the talk as follows, depending on the allowed time:

- Introduction of the probabilistic λ-calculus and of its semantics,
- Introduction of the system of monadic affine sized types for AST termination,
- Necessity of affinity for soundness on an example,
- If time allows, main elements of the proof of soundness of the type system: reducibility candidates parameterized by a probability, continuity property on these sets, relation between these sets and sized walks, treatment of the letrec construct in the reducibility proof.

- Introduction of PTS and of the system with dependent types ; main differences with the previous system
- If time allows, we give the relation between PTS and reducibility candidates which makes the reducibility proof work.

References

[1] Gilles Barthe, Maria João Frade, Eduardo Giménez, Luis Pinto & Tarmo Uustalu (2004): *Type-based termination of recursive definitions*. *MSCS* 14(1), pp. 97–141, doi:10.1017/S0960129503004122.

[2] Gilles Barthe, Benjamin Grégoire & Santiago Zanella Béguelin (2009): *Formal certification of code-based cryptographic proofs*. In Zhong Shao & Benjamin C. Pierce, editors: *POPL 2009*, ACM, pp. 90–101.

[3] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud & Santiago Zanella Béguelin (2011): *Computer-Aided Security Proofs for the Working Cryptographer*. In Phillip Rogaway, editor: *CRYPTO 2011*, *LNCS* 6841, Springer, pp. 71–90. Available at `http://dx.doi.org/10.1007/978-3-642-22792-9_5`.

[4] Olivier Bournez & Claude Kirchner (2002): *Probabilistic Rewrite Strategies. Applications to ELAN*. In Sophie Tison, editor: *RTA 2002*, *LNCS* 2378, Springer, pp. 252–266. Available at `http://dx.doi.org/10.1007/3-540-45610-4_18`.

[5] Aleksandar Chakarov & Sriram Sankaranarayanan (2013): *Probabilistic Program Analysis with Martingales*. In Natasha Sharygina & Helmut Veith, editors: *CAV 2013*, *LNCS* 8044, Springer, pp. 511–526. Available at `http://dx.doi.org/10.1007/978-3-642-39799-8_34`.

[6] Krishnendu Chatterjee, Hongfei Fu & Amir Kafshdar Goharshady (2016): *Termination Analysis of Probabilistic Programs Through Positivstellensatz's*. In Swarat Chaudhuri & Azadeh Farzan, editors: *CAV 2016*, *LNCS* 9779, Springer, pp. 3–22. Available at `http://dx.doi.org/10.1007/978-3-319-41528-4_1`.

[7] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný & Rouzbeh Hasheminezhad (2016): *Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs*. In Rastislav Bodík & Rupak Majumdar, editors: *POPL 2016*, ACM, pp. 327–342, doi:10.1145/2837614.2837639. Available at `http://dl.acm.org/citation.cfm?id=2837614`.

[8] Ugo Dal Lago & Charles Grellois (2016): *Probabilistic Termination by Monadic Affine Sized Typing (Long Version)*. `https://arxiv.org/abs/1701.04089`.

[9] Javier Esparza, Andreas Gaiser & Stefan Kiefer (2012): *Proving Termination of Probabilistic Programs Using Patterns*. In P. Madhusudan & Sanjit A. Seshia, editors: *CAV 2012*, *LNCS* 7358, Springer, pp. 123–138, doi:10.1007/978-3-642-31424-7.

[10] Luis María Ferrer Fioriti & Holger Hermanns (2015): *Probabilistic Termination: Soundness, Completeness, and Compositionality*. In Sriram K. Rajamani & David Walker, editors: *POPL 2015*, ACM, pp. 489–501, doi:10.1145/2676726.2677001. Available at `http://dl.acm.org/citation.cfm?id=2676726`.

[11] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz & Joshua B. Tenenbaum (2008): *Church: a language for generative models*. In David A. McAllester & Petri Myllymäki, editors: *UAI 2008*, AUAI Press, pp. 220–229.

[12] Martin Hofmann (1997): *A Mixed Modal/Linear Lambda Calculus with Applications to Bellantoni-Cook Safe Recursion*. In Mogens Nielsen & Wolfgang Thomas, editors: *CSL '97*, *LNCS* 1414, Springer, pp. 275–294.

[13] John Hughes, Lars Pareto & Amr Sabry (1996): *Proving the Correctness of Reactive Systems Using Sized Types*. In Hans-Juergen Boehm & Guy L. Steele Jr., editors: *POPL'96*, ACM Press, pp. 410–423, doi:10.1145/237721.240882. Available at `http://dl.acm.org/citation.cfm?id=237721`.

[14] Ugo Dal Lago & Charles Grellois (2017): *Probabilistic Termination by Monadic Affine Sized Typing*. In Hongseok Yang, editor: *ESOP 2017*, *Lecture Notes in Computer Science* 10201, pp. 393–419. Available at `http://dx.doi.org/10.1007/978-3-662-54434-1_15`.

[15] Karel de Leeuw, Edward F. Moore, Claude E. Shannon & Norman Shapiro (1956): *Computability by probabilistic machines*. Automata Studies 34, pp. 183–212.

[16] Christopher D. Manning & Hinrich Schütze (2001): *Foundations of statistical natural language processing*. MIT Press.

[17] Rajeev Motwani & Prabhakar Raghavan (1995): *Randomized Algorithms*. Cambridge University Press, doi:10.1017/CBO9780511814075.

[18] Judea Pearl (1989): *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning.

[19] Sebastian Thrun (2002): *Robotic Mapping: A Survey*. In: *Exploring Artificial Intelligence in the New Millenium*, Morgan Kaufmann.

[20] David Tolpin, Jan Willem van de Meent, Hongseok Yang & Frank Wood (2016): *Design and Implementation of Probabilistic Programming Language Anglican*. arXiv preprint *arXiv:1608.05263*.

[21] Hongwei Xi (2002): *Dependent Types for Program Termination Verification*. Higher-Order and Symbolic Computation 15(1), pp. 91–131, doi:10.1023/A:1019916231463.