Probabilistic Programming: A True Verification Challenge

Joost-Pieter Katoen

Software Modelling and Verification, RWTH Aachen University, Germany
Formal Methods and Tools, University of Twente, The Netherlands
katoen@cs.rwth-aachen.de

Probabilistic programming. Probabilistic programs [6] are sequential programs, written in languages like `C`, `Java`, `Scala`, or `ML`, with two added constructs: (1) the ability to draw values at random from probability distributions, and (2) the ability to condition values of variables in a program through observations. For a comprehensive treatment, see [3]. They have a wide range of applications. Probabilistic programming is at the heart of machine learning for describing distribution functions; Bayesian inference is pivotal in their analysis. Probabilistic programs are central in security for describing cryptographic constructions (such as randomised encryption) and security experiments. In addition, probabilistic programs are an active research topic in quantitative information flow. Quantum programs are inherently probabilistic due to the random outcomes of quantum measurements. Finally, probabilistic programs can be used for approximate computing, e.g., by specifying reliability requirements for programs that allocate data in unreliable memory and use unreliable operations in hardware (so as to save energy dissipation) [1]. Other applications include [4] scientific modeling, information retrieval, bio–informatics, epidemiology, vision, seismic analysis, semantic web, business intelligence, human cognition, and more. Microsoft has started an initiative to improve the usability of probabilistic programming which has resulted in languages such as `R2` [13] and `Tabular` [5] emerged.

What is special about probabilistic programs? They are typically small (up to a few hundred lines), but hard to understand and analyse, let alone algorithmically. For instance, the elementary question of almost-sure termination—for a given input, does a probabilistic program terminate with probability one?—is as hard as the universal halting problem—does an ordinary program halt on all possible inputs? [11]. Loop invariants of probabilistic programs typically involve quantitative statements and synthesizing them requires more involved techniques than for ordinary programs [12]. As a final indication of their complexity, we mention that probabilistic programs allow to draw values from parametric probability distributions. Obtaining quantitative statements such as "what is the expected value of program variable `x` on termination?" require non-trivial reasoning about such parametric distributions.

Analysing probabilistic programs. Bugs easily occur. We develop program analysis techniques, based on static program analysis, deductive verification, and model checking, to make probabilistic programming more reliable, i.e., less buggy. Starting from a profound understanding from the intricate semantics of probabilistic programs (including features such as observations, possibly diverging loops, continuous variables, non-determinism, as well as unbounded recursion), we study fundamental problems such as checking program equivalence, loop-invariant synthesis, almost-sure termination, and pre- and postcondition reasoning. The aim is to study the computational hardness of these problems as well as to develop (semi-) algorithms and accompanying tool-support. The ultimate goal is to provide lightweight automated means to the probabilistic programmer so as check elementary program properties.

Formal semantics and verification. In this invited talk, I will survey recent progress on the formal semantics and verification of (parametric) probabilistic programs. This involves relating weakest pre-conditions and operational semantics [9, 7], loop-invariant synthesis techniques by constraint solving [12, 8], hardness results on almost-sure termination [11], and verifying parametric probabilistic models and their applications [2, 10, 14].

Acknowledgement. This work is funded by the EU FP7-projects SENSATION and MEALS, and the Excellence Initiative of the German federal and state government.

References

1. M. Carbin, S. Misailovic, and M. C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *Proc. of OOPSLA*, pages 33–52. ACM Press, 2013.
2. C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám. PROPhESY: A PRObabilistic ParamEter SYnthesis tool. In *Proc. of CAV*, volume 9206 of *LNCS*. Springer, 2015.
3. N. D. Goodman and A. Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages.* (electronic), 2014. http://dippl.org.
4. A. D. Gordon. An agenda for probabilistic programming: Usable, portable, and ubiquitous, 2013.
5. A. D. Gordon, T. Graepel, N. Rolland, C. V. Russo, J. Borgström, and J. Guiver. Tabular: a schema-driven probabilistic programming language. In *Proc. of POPL*, pages 321–334. ACM Press, 2014.
6. A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Proc. of FOSE*, pages 167–181. ACM Press, 2014.
7. F. Gretz, N. Jansen, B. L. Kaminski, J.-P. Katoen, A. McIver, and F. Olmedo. Conditioning in probabilistic programming. In *Proc. of MFPS*, ENTCS.
8. F. Gretz, J.-P. Katoen, and A. McIver. PRINSYS - on a quest for probabilistic loop invariants. In *Proc. of QEST*, volume 8054 of *LNCS*, pages 193–208. Springer, 2013.
9. F. Gretz, J.-P. Katoen, and A. McIver. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Perform. Eval.*, 73:110–132, 2014.

10. N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen, and B. Becker. Accelerating parametric probabilistic verification. In *Proc. of QEST*, volume 8657 of *LNCS*, pages 404–420. Springer, 2014.
11. B. L. Kaminski and J.-P. Katoen. On the hardness of almost-sure termination. In *Proc. of MFCS*, volume 9234 of *LNCS*. Springer, 2015.
12. J.-P. Katoen, A. McIver, L. Meinicke, and C. C. Morgan. Linear-invariant generation for probabilistic programs. In *Proc. of SAS*, volume 6337 of *LNCS*, pages 390–406. Springer, 2010.
13. A. V. Nori, C.-K. Hur, S. K. Rajamani, and S. Samuel. R2: An efficient MCMC sampler for probabilistic programs. In *Proc. of AAAI*. AAAI Press, 2014.
14. S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella, and J.-P. Katoen. A greedy approach for the efficient repair of stochastic models. In *Proc. of NFM*, volume 9058 of *LNCS*, pages 295–309. Springer, 2015.