Exact Probabilistic Inference Using Generating Functions*

Lutz Klinkenberg, Tobias Winkler, Mingshuai Chen, and Joost-Pieter Katoen

RWTH Aachen University, Aachen, Germany
{lutz.klinkenberg,tobias.winkler,chenms,katoen}@cs.rwth-aachen.de

Probabilistic programs are typically normal-looking programs describing posterior probability distributions. They intrinsically code up randomized algorithms and have long been at the heart of modern machine learning and approximate computing. We explore the theory of *generating functions* [19] and investigate its usage in the exact quantitative reasoning of probabilistic programs. Important topics include the exact representation of program semantics [13], proving exact program equivalence [5], and – as our main focus in this extended abstract – exact probabilistic inference.

In probabilistic programming, *inference* aims to derive a program's posterior distribution. In contrast to approximate inference, inferring *exact* distributions comes with several benefits [8], e.g., no loss of precision, natural support for symbolic parameters, and efficiency on models with certain structures. Exact probabilistic inference, however, is a notoriously hard task [6,12,17,18]. The challenges mainly arise from three program constructs: (1) unbounded while-loops and/or recursion, (2) infinite-support distributions, and (3) conditioning (via posterior observations). We present our ongoing research in addressing these challenges (with a focus on conditioning) leveraging generating functions and show their potential in facilitating exact probabilistic inference for discrete probabilistic programs.

1 Inference in Probabilistic Programs

State-of-the-Art. Most existing probabilistic programming languages implement *sampling*-based inference algorithms rooted in the principles of Monte Carlo [15], thereby yielding numerical approximations of the exact results, see, e.g., [9]. In terms of semantics, many probabilistic systems employ *probability density function* (PDF) representations of distributions, e.g., (λ)PSI [7,8], AQUA [11], HAKARU [16], and the density compiler in [3,4]. These systems are dedicated to inference (with conditioning) for programs encoding (joint discrete-)continuous distributions. Reasoning about the underlying PDF representations, however, amounts to resolving complex integral expressions in order to answer inference queries, thus confining these techniques either to (semi-)numerical methods [3,4,11,16] or exact methods yet limited to bounded looping behaviors [7,8]. DICE [10] employs weighted model counting to enable exact inference for discrete probabilistic programs, yet is also confined to statically bounded loops. The tool MORA [1,2] supports exact inference for various types of Bayesian networks, but relies on a restricted form of intermediate representation known as prob-solvable loops.

The PGF Approach. Klinkenberg et al. [13] provide a program semantics that allows for exact quantitative reasoning about probabilistic programs without conditioning. They exploit a denotational approach à la Kozen [14] and treat a probabilistic program as a *distribution transformer*, i.e., mapping a distribution over the inputs (the prior) into a distribution after execution of the program (the posterior). In [13], the domain of discrete distributions is represented in terms of *probability generating functions* (PGFs), which are a special kind of generating functions [19]. This representation comes with several benefits: (a) it naturally encodes common, infinite-support distributions (and variations thereof) like the geometric or Poisson distribution in compact, *closed-form* representations; (b) it allows for compositional reasoning and, in particular, in contrast to representations in terms of density or mass functions, the effective computation of (high-order) moments; (c) tail bounds, concentration bounds, and other properties of interest can be extracted with relative ease from a PGF; and (d) expressions containing parameters, both for probabilities and for assigning new values to program variables, are naturally supported. Some successfully implemented ideas based on PGFs, e.g., for deciding probabilistic equivalence and for proving

* Extended abstract accepted by LAFI 2023 – the Languages for Inference workshop co-located with POPL 2023.

non-almost-sure termination, are presented in [5,13], which address especially the aforementioned challenges (1) and (2) for exact probabilistic inference *without* conditioning.

2 Taming Conditioning Using PGFs

The creation of generative models is a challenging task, as these models oftentimes need expert domain knowledge. Therefore, the concept of *conditioning as a first-class language element* is crucial as it allows for a natural and intuitive approach to the creation of models. Our current research aims to *extend the PGF approach towards exact inference for probabilistic programs with conditioning – thus addressing challenges (1), (2), and (3) – and to push the limits of automation as far as possible*. To this end, we are in the process of developing an exact, symbolic inference engine based on the open-source, PGF-based tool PRODIGY [5]. We illustrate below its current capability to cater for conditioning via two examples.

$\{w \coloneqq 0\}[5/7]\{w \coloneqq 1\}$;

if$(w = 0)\{\ c \coloneqq \texttt{poisson}(6)\ \}$
else $\{\ c \coloneqq \texttt{poisson}(2)\ \}$;

observe$(c = 5)$

Prog. 1: The telephone operator.

$x \coloneqq 1$;

while$(x = 1)\{$
$\quad\{c \coloneqq c + 1\}[1/2]\{x \coloneqq 0\}$; $\}$

observe$(c \equiv 1 \pmod 2)$

Prog. 2: The odd geometric distribution.

Conditioning in Loop-Free Programs. Prog. 1 is a loop-free probabilistic program encoding an infinite-support distribution. It describes a telephone operator who is unaware of whether today is a weekday or weekend. The operator's initial belief is that with probability $5/7$ it is a weekday ($w = 0$) and thus with probability $2/7$ weekend ($w = 1$). Usually, on weekdays there are 6 incoming calls per hour on average; on weekends this rate decreases to 2 calls – both rates are subject to a Poisson distribution. The operator observes 5 calls in the last hour. The inference task is to compute the distribution in which the initial belief is updated based on the posterior observation. PRODIGY can automatically infer that $\Pr(w = 0) = \frac{1215}{1215+2\cdot e^4} \approx 0.9178$.

Conditioning Outside of Loops. Prog. 2 describes an iterative algorithm that repeatedly flips a fair coin – while counting the number of trials – until seeing heads, and observes that this number is odd. Whereas Prog. 1 can be handled by (λ)PSI [7,8], Prog. 2 cannot, as (λ)PSI do not support programs with unbounded looping behaviors. However, given a suitable invariant as described in [5], PRODIGY is able to reason about the posterior distribution of Prog. 2 in an automated fashion using the *second-order PGF* (SOP) technique [5]: the resulting posterior distribution for any input with $c = 0$ is $\frac{3\cdot c}{(4-c^2)}$ which encodes precisely a closed-form solution for the generating function $\sum_{n=0}^{\infty} -3 \cdot c^n \cdot \left(2^{-2-n} \cdot (-1 + (-1)^n)\right)$.

3 Future Directions

A natural question is whether we can tackle exact inference when conditioning occurs *inside* of a loop. As argued in [17], more advanced inference techniques are required to answer this question. In fact, to the best of our knowledge, there is no (semi-)automated exact inference technique that allows for the presence of observe statements inside a (possibly unbounded) loop (an exception could be the potentially automatable conditional weakest preexpectation calculus [17]). This is precisely our current research focus. One promising idea is to develop a non-trivial syntactic restriction of the programming language, where the more advanced SOP technique [5] can be generalized to address conditioning inside loops.

The possibility to incorporate symbolic parameters in PGF representations can enable the application of well-established optimization methods, e.g., maximum-likelihood estimations and parameter fitting, to the inference for probabilistic programs. Other interesting future directions include deciding equivalence of probabilistic programs with conditioning, amending our method to continuous distributions using characteristic functions, and exploring the potential of PGFs in differentiable programming.

References

1. Ezio Bartocci, Laura Kovács, and Miroslav Stankovic. Analysis of Bayesian networks via prob-solvable loops. In *ICTAC*, volume 12545 of *LNCS*, pages 221–241. Springer, 2020.
2. Ezio Bartocci, Laura Kovács, and Miroslav Stankovic. MORA - Automatic generation of moment-based invariants. In *TACAS (1)*, volume 12078 of *LNCS*, pages 492–498. Springer, 2020.
3. Sooraj Bhat, Ashish Agarwal, Richard W. Vuduc, and Alexander G. Gray. A type theory for probability density functions. In *POPL*, pages 545–556. ACM, 2012.
4. Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio V. Russo. Deriving probability density functions from probabilistic functional programs. *Log. Methods Comput. Sci.*, 13(2), 2017.
5. Mingshuai Chen, Joost-Pieter Katoen, Lutz Klinkenberg, and Tobias Winkler. Does a program yield the right distribution? Verifying probabilistic programs via generating functions. In *CAV (1)*, volume 13371 of *LNCS*, pages 79–101. Springer, 2022.
6. Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intell.*, 42(2-3):393–405, 1990.
7. Timon Gehr, Sasa Misailovic, and Martin T. Vechev. PSI: Exact symbolic inference for probabilistic programs. In *CAV (1)*, volume 9779 of *LNCS*, pages 62–83. Springer, 2016.
8. Timon Gehr, Samuel Steffen, and Martin T. Vechev. λPSI: Exact inference for higher-order probabilistic programs. In *PLDI*, pages 883–897. ACM, 2020.
9. Bradley Gram-Hansen. *Extending probabilistic programming systems and applying them to real-world simulators*. PhD thesis, University of Oxford, 2021.
10. Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.*, 4(OOPSLA):140:1–140:31, 2020.
11. Zixin Huang, Saikat Dutta, and Sasa Misailovic. AQUA: Automated quantized inference for probabilistic programs. In *ATVA*, volume 12971 of *LNCS*, pages 229–246. Springer, 2021.
12. Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the hardness of analyzing probabilistic programs. *Acta Inform.*, 56(3):255–285, 2019.
13. Lutz Klinkenberg, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Joshua Moerman, and Tobias Winkler. Generating functions for probabilistic programs. In *LOPSTR*, volume 12561 of *LNCS*, pages 231–248. Springer, 2020.
14. Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981.
15. Nicholas Metropolis and Stanisław Ulam. The Monte Carlo method. *J. Am. Stat. Assoc.*, 44(247):335–341, 1949.
16. Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In *FLOPS*, volume 9613 of *LNCS*, pages 62–79. Springer, 2016.
17. Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle McIver. Conditioning in probabilistic programming. *ACM Trans. Program. Lang. Syst.*, 40(1):4:1–4:50, 2018.
18. Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1):273–302, 1996.
19. Herbert S Wilf. *Generatingfunctionology*. CRC press, 2005.