A Formalization of Probabilistic Programs in First-Order Logic

Pritom Rajkhowa

May 2019

1 Introduction

The seminal paper on probabilistic algorithms Michael Rabin [1] has enabled the recognized of the substantial benefits of the introduction of randomization in designing algorithms which are widely employed in cryptographic/privacy protocols, stochastic games, randomized algorithms [2], several emerging applications in the areas of machine learning and AI, etc. These programs are notoriously difficult to get right with a certain probability as well as reasoning about the correctness because probabilistic programs (PPs) adds the following additional complexity and challenges compared with deterministic programs:

- Although these programs are typically just a few numbers of lines, they are hard to understand and analyze, let alone algorithmically.
- The termination of these program for a given input is as hard as the universal halting problem [3].
- The correctness of loops in such programs can be proved by inferring special expectations called the quantitative loop invariants which require more involved techniques for synthesizing compared to their deterministic counterparts [4, 5, 6].
- Another issue associated with such programs arises from introducing randomness into it that variables cannot be viewed as single values; they must be viewed as distributions. Whereas existing approaches

such as [4, 5, 6] usually take into consideration only upper and lower bounds over program variables or expected values. Such information may be insufficient to characterize the distributions of variables as highlighted in [7]. Similarly, the (co-)variances and other higher-order moments of variables are also needed. Recently published work [8], tried to address these issues. However, they do not address loop conditions, nested loops, and multipath loops, while our work in this paper addresses these as well.

The main challenge when reasoning about correctness for PPs with loops rely on what has been known as quantitative invariants [4]. Some of the earlier approaches [4, 9] use weakest pre-expectations to generate quantitative invariants. PRINSYS [5] computed quantitative invariants using a constraint-based method where a loop is annotated with a template invariant. Other methods [10, 11] use approaches where the expectations of expressions over program variables remain invariant. The major drawback of these approaches, called martingale-based approaches, is that they are not fully automatic, unlike our proposed approach. These methods require templates and hints as user intervention to generate quantitative invariants. Another major drawback of these approaches is that the generated invariants consider only expected values [10], and those generated invariants cannot be used to compute higher-order moments to describe the distribution generated by the PP. A fully automated system is described in [8] to generate quantitative invariants which can compute expected values of the variables and their corresponding higher-order moment's values. But it can handle only a set of syntactically restricted programs with loop. The limitation of this approach [8] is that it cannot handle PPs with multiple-path body presented in Table **??** (a),(b).

In this paper, we describe a system for automatically computing any higher-order moment of the avariables generated by the PPs. It translates programs to first-order logic with quantifiers on natural numbers, and the extension of Lin's [12] proposal for translating a deterministic non-concurrent program. Lin's approach handles loop by systematically encoding iterations by inductive definitions. The termination of those loops is axiomatized with constraints on specially introduced constants. Then those generated axioms are simplified by eliminating probabilistic choices using statistical properties and compute the closed-form solutions of inductive definitions using a dedicated recurrence solver (RS - solver).

Currently, our system accepts programs written with assignments, sequences, conditionals, and while loops. The variables can have any type - as long as there is a way to reason about the values of this type. We tested our system on a suite of benchmarks [10, 8] that include many well-known algorithms like a different variant of Coupon Collector's, and Random Walk algorithms, Sum of Random Series game, Product of Dependents Random Variables games and Modeling Binomial distribution. Like existing work [8], our system is fully automatic and compute the k^{th}-moments of the variables for a given integer $k \geq 1$ for input PP. It also supports PPs with given parametrized distributions. But the essential difference between the two approaches is that our approach precisely summarizes the semantics of general loops. In contrast, [8] is based on recurrences analysis which focuses on accurate computing information about the syntactically restricted loop. In addition to that our work also focuses on the problem such as computing termination probabilities and runtime complexity.

2 Motivating Examples

2.1 Example 1

We first introduce probabilistic programs with an example. Then demonstrate how the proposed approach works.

Consider the following program probabilistic program. This program represents a game where a player flips a coin for y times. The player wins k dollars if the head turns up at the k^{th} flip. The following program models this game when the head probability of the coin is 0.5:

```
x=0; while (0<y) { x = x + y [1/2] x; y = y - 1;}
```

Our translator would be translated to a set of axioms $\Pi_P^{\vec{X}}$ like the following where variables $\vec{X} = \{x, y\}$

$$
\begin{aligned}
&x_1 = x_3(N_1), y_1 = y_3(N_1),\\
&\forall n_1.y_3(n_1 + 1) = y_3(n_1) - 1,\\
&\forall n_1.x_3(n_1 + 1) = 0.5 \times (x_3(n_1) + y_3(n_1)) + (1 - 0.5) \times x_3(n_1),\\
&x_3(0) = 0, y_3(0) = y\\
&\neg(y_3(N_1) > 0),\\
&\forall n_1.n_1 < N_1 \rightarrow (y_3(n_1) > 0).
\end{aligned}
$$

where x_1 and y_1 denote the output values of x and y, respectively, $x_3(n_1)$ and $y_3(n_1)$ the values of x and y during the n_1th iteration of the loop, respectively. Also, N_1 is a system-generated natural number constant denoting the number of iterations that the outer loop runs before exiting. The expected amount of money the player can win from this game is $E[x_1] = E[x_3(N_1)]$.

By using RS, our translator computes closed-form solution for $x_3(n_1 + 1)$ and moment based closed-form solution $E[y_3(n_1 + 1)]$, eliminates them, and produces the following axioms:

$$
\begin{aligned}
&E[x_1] = \frac{2 \times y \times N_1 - N_1^2 - N_1}{4}, y_1 = y - N_1, \\
&\neg((y - N_1) > 0), \\
&\forall n_1 . n_1 < N_1 \rightarrow (y - n_1) > 0.
\end{aligned}
$$

From these axioms, our system can show the post expectation of the above algorithm that expected the amount of the money a player can win is $E[x_1] = \frac{y^2-y}{4}$ with respect to pre-expectation $y > 0$. Our system can also show the post-expectation the program for the higher moment such as $E[x_1^2] = \frac{y^2-y}{4}$ and $E[x_1^3] = \frac{y^2-y}{4}$.

2.2 Example 2

We illustrate how our approach effectively handle nested loop where other approaches failed using the above simple example. The C code snippet P is from the benchmark used in [13] with variables $\vec{X} = \{x, y, k, L, M\}$. Our translator would be translated to a set of axioms $\Pi_P^{\vec{X}}$ like the following:

```
real x, y;
int k = 0 ;
while(x<=L)
{
    y = 0;
    while(y<=M){
        y = y + UniReal(-0.1,0.2);
    }
    x = x + UniReal(-0.1,0.2);
    k = k+1;
}
```

Our translator would be translated to a set of axioms $\Pi_P^{\vec{X}}$ like the following:

$$
\begin{aligned}
&x_1 = x_6(N_2), y_1 = y_6(N_2), k_1 = k_6(N_2), M_1 = M, L_1 = L,\\
&\forall n_1, n_2.y_2(n_1 + 1, n_2) = y_2(n_1, n_2) + uniReal(-0.1, 0.2, n_1, n_2)\\
&\forall n_2.y_2(0, n_2) = 0,\\
&\forall n_1, n_2.\neg(y_3(N_1(n_2), n_2) < M),\\
&\forall n_1, n_2.n_1 < N_1(n_2) \rightarrow (y_3(n_1, n_2) < M),\\
&\forall n_2.y_6(n_2 + 1) = y_2(N_1(n_2), n_2),\\
&\forall n_2.x_6(n_2 + 1) = x_6(n_2) + uniReal(-0.1, 0.2, n_2)\\
&\forall n_2.k_6(n_2 + 1) = k_6(n_2) + 1\\
&x_6(0) = y, y_6(0) = 0, k_6(0) = 0\\
&\neg(x_6(N_2) < L),\\
&\forall n_2.n_2 < N_2 \rightarrow (x_6(N_2) < L)
\end{aligned}
$$

where x_1, y_1, k_1, M_1 and N_1 denote the output values of x, y, k, M and N, respectively, $y_2(n_1, n_2)$ the value of y at n_1th iteration of the inter loop during n_2th iteration of the outer loop. $x_6(n_2)$, $y_6(n_2)$ and $k_6(n_2)$ the values of x, y and k during the n_2th iteration of the loop, respectively. Also N_2 is a system generated natural number constant denoting the number of iterations that the outer loop runs before exiting; N_1 is a system generated natural number function denoting that for each n_2, $N_1(n_2)$ is the number of iterations that the inner loop runs during the n_2th iteration of the outer loop.

It tries to find the closed form solutions of moment base recurrences relations generated by translator using our desgined system recurrence solver module(RS). After substituting moment base recurrences relations closed-form solution with resulted set of axoims $\Pi'^{\vec{X}}_P$ are as follows:

$$
\begin{aligned}
&E[x_1] = \frac{10 \times N_2}{3}, E[y_1] = \frac{10 \times N_1(N_2)}{3}, E[k_1] = N_2, , M_1 = M, L_1 = L\\
&\forall n_2.\neg(\frac{10 \times N_1(n_2)}{3} < M), \forall n_1, n_2.n_1 < N_1(n_2) \rightarrow (\frac{10 \times N_1(n_2)}{3} < M),\\
&\neg(\frac{10 \times N_2}{3} < L), \forall n_2.n_2 < N_2 \rightarrow (\frac{10 \times N_2}{3}) < L)
\end{aligned}
$$

For the above program, our system can show the post expectation of variable x and y as $\frac{10 \times N_2}{3}$ and $\frac{10 \times N_1(N_2)}{3}$ respectively.

When it tries to find the closed form solutions of higher moment or mixed moment using our desgined system recurrence solver module(RS). After substituting higher moment or mixed moment closed-form solution with resulted set of axoims $\Pi'^{\vec{X}}_{P}$ are as follows:

$$
\begin{aligned}
&Var[x_1] = \frac{9 \times N_1^4}{16} - \frac{9 \times N_1^3}{8} + \frac{15 \times N_1^2}{16} - \frac{3 \times N_1}{2} + 1, \\
&Var[y_1] = \frac{9 \times N_1^4}{16} - \frac{9 \times N_1^3}{8} + \frac{33 \times N_1^2}{16} - \frac{3 \times N_1}{2} + 1, \\
&Var[f_1] = \frac{9}{16}, \\
&\neg((\frac{9 \times N_1^4}{16} - \frac{9 \times N_1^3}{8} + \frac{15 \times N_1^2}{16} - \frac{3 \times N_1}{2} + 1) < M), \\
&\forall n_1 . n_1 < N_1 \rightarrow ((\frac{9 \times n_1^4}{16} - \frac{9 \times n_1^3}{8} + \frac{15 \times n_1^2}{16} - \frac{3 \times n_1}{2} + 1) < M).
\end{aligned}
$$

References

[1] M. O. Rabin, "Probabilistic algorithms in finite fields," *SIAM J. Comput.*, vol. 9, no. 2, pp. 273–280, 1980. [Online]. Available: https://doi.org/10.1137/0209024

[2] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, 2015. [Online]. Available: https://doi.org/10.1038/nature14541

[3] B. L. Kaminski and J. Katoen, "On the hardness of almost-sure termination," *CoRR*, vol. abs/1506.01930, 2015. [Online]. Available: http://arxiv.org/abs/1506.01930

[4] J.-P. Katoen, A. K. McIver, L. A. Meinicke, and C. C. Morgan, "Linear-invariant generation for probabilistic programs:," in *Static Analysis*, R. Cousot and M. Martel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 390–406.

[5] F. Gretz, J.-P. Katoen, and A. McIver, "Prinsys—on a quest for probabilistic loop invariants," in *Quantitative Evaluation of Systems*, K. Joshi, M. Siegle, M. Stoelinga, and P. R. D'Argenio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 193–208.

[6] A. Chakarov and S. Sankaranarayanan, "Expectation invariants for probabilistic program loops as fixed points," in *Static Analysis*, M. Müller-Olm and H. Seidl, Eds. Cham: Springer International Publishing, 2014, pp. 85–100.

[7] P. L. Novi Inverardi and A. Tagliani, "Discrete distributions from moment generating function," *Appl. Math. Comput.*, vol. 182, no. 1, pp. 200–209, Nov. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.amc.2006.03.048

[8] E. Bartocci, L. Kovács, and M. Stankovic, "Automatic generation of moment-based invariants for prob-solvable loops," *CoRR*, vol. abs/1905.02835, 2019. [Online]. Available: http://arxiv.org/abs/1905.02835

[9] A. McIver and C. Morgan, *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. SpringerVerlag, 2004.

[10] S. Kura, N. Urabe, and I. Hasuo, "Tail probabilities for randomized program runtimes via martingales for higher moments," *CoRR*, vol. abs/1811.06779, 2018. [Online]. Available: http://arxiv.org/abs/1811.06779

[11] G. Barthe, T. Espitau, L. M. F. Fioriti, and J. Hsu, "Synthesizing probabilistic invariants via doob's decomposition," *CoRR*, vol. abs/1605.02765, 2016. [Online]. Available: http://arxiv.org/abs/1605.02765

[12] F. Lin, "A formalization of programs in first-order logic with a discrete linear order," *Artificial Intelligence*, vol. 235, pp. 1 – 25, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437021630011X

[13] K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad, "Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs," *CoRR*, vol. abs/1510.08517, 2015. [Online]. Available: http://arxiv.org/abs/1510.08517