PROPER

a Tool for Analyzing Termination and Assertions for Probabilistic Programs

Anonymous Author(s)

ABSTRACT

Probabilistic programs combine probabilistic reasoning models with Turing complete programming languages, unify formal descriptions of calculation and uncertain knowledge, and can effectively deal with complex relational models and uncertain problems. This paper presents PROPER, a tool for analyzing termination and assertions for affine probabilistic programs. On one hand, it can help to analyze the termination property of affine probabilistic programs both qualitatively and quantitatively. It can check whether a probabilistic program terminates with probability 1, estimate the upper bound of expected termination time, and calculate the number of steps after which the termination probability of the given probabilistic program decreases exponentially. On the other hand, it can estimate the correct probability interval for a given assertion to hold, which helps to analyze the influence of uncertainty of variables on the results of probabilistic programs. The effectiveness of PROPER is demonstrated through various affine probabilistic programs.

CCS CONCEPTS

• **Software and its engineering → Software notatins and tools**;

KEYWORDS

Probabilistic Programming, Program Verification, Termination, Ranking Supermartingale

ACM Reference Format:
Anonymous Author(s). 2020. PROPER: a Tool for Analyzing Termination and Assertions for Probabilistic Programs. In *Proceedings of The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

There are a large number of uncertainty problems in the real world, which require us to use prerequisite knowledge and deductive reasoning to predict the results, that is to say, to make decisions on nondeterministic problems through probabilistic reasoning. Therefore, probabilistic programs are put forward for that purpose. Probabilistic programs are a kind of logic programs with probabilistic facts. They make probabilistic reasoning models easier to build and can estimate the possibilities of certain events to occur. Probabilistic programs have a wide range of applications in various fields such as business, military, scientific research and daily life. Probability is becoming more and more important in actual calculations, such as risk analysis, medical decision-making, differential privacy mechanisms, etc. The analysis of probabilistic programs has also received widespread attention in academia and industry.

In the current work we present PROPER, a tool for probabilistic program verification, namely, analyzing termination and assertions for affine probabilistic programs (probabilistic programs whose all arithmetic expressions are linear). The tool builds on [5, 13, 15]. It can analyze the termination property of a given affine probabilistic program qualitatively and quantitatively. The qualitative analysis asks whether an affine program terminates with probability 1 (almost-sure termination). The quantitative analysis aims to approximate the expected termination time (expectation problem) and compute a bound N such that the probability to terminate after N steps decreases exponentially (concentration problem). It can also estimate the correct probability interval for an affine probabilistic program to conform to a specification given as an assertion. More specifically, PROPER provides the following functionalities:

(1) Defining and parsing probabilistic programs. More than ten kinds of probability distributions are built in, which is convenient to be called to build probabilistic models.
(2) Reducing the termination analysis of a given program to a linear programming problem. We also consider the concentration results under the premise that the program is terminating.
(3) Reducing the estimation of a given assertion to a polyhedron solving problem. In particular, it can handle probabilistic programs with infinitely many states.

The tool and the experimental data given in this paper are all available in the repository https://github.com/Healing1219/PROPER.

2 PROBABILISTIC PROGRAMS

In order to analyze and verify probabilistic programs, we first need a probabilistic language sufficiently expressive and easy to understand. We follow [15] to define a probabilistic language, whose syntax specification is shown in Figure 1. The statements in the probabilistic language are similar to those in classic imperative languages, mainly composed of three types of statements: assignment, condition-branch (if-else) and loop statements (while). The main difference is that we now have a collection of random value generators (discrete distribution, Binomial distribution, Poisson distribution, Integer Uniform distribution, Real Uniform distribution, Exponential distribution, Gamma distribution, Beta distribution, Geometric distribution), which can be used to simulate different probability distributions. Variables are classified into two types: program variables and sampling variables. Program variables include integer, real, and boolean variables. Boolean variables are mainly used for condition and loop statements. Sampling variables are assigned with dynamically generated values when the program

ESEC/FSE 2020, 8 - 13 November, 2020, Sacramento, California, United States
© 2020 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

```
     program  :=  typeSpecifier main{stmt*}
        stmt  :=  assign | condStmt | while
      assign  :=  intAssign | realAssign
    condStmt  :=  ifStmt | ifElseStmt
       while  :=  while (test) stmt*
   intAssign  :=  intVar = intConst | intVar ~ intRandom
  realAssign  :=  realVar = realConst | realVar ~ realRandom
   intRandom  :=  uniformInt(intConst, intConst)
                  | Bernoulli(intConst, intConst)
                  . . .
  realRandom  :=  uniformReal(realConst, realConst)
                  | Gaussian(realConst, realConst)
                  . . .
     intExpr  :=  intConst | intRandom | intExpr ± intExpr
                  intConst* intExpr | intExpr / intConst
    realExpr  :=  realConst | realRandom | realExpr ± realExpr
                  realConst* realExpr | realExpr / realConst
    boolExpr  :=  true | false | boolExpr ∧ boolExpr
                  intExpr relop intExpr | realExpr relop realExpr
       relop  :=  < | > | ≥ | ≤ | ==
```

Figure 1: Syntax specification of a probabilistic language

is running, which is subject to a continuous or discrete probability distribution.

We use control flow graphs (CFGs) to express the semantics of probabilistic programs. Formally, a CFG is a tuple in the form $(L, X, R, \mapsto, \perp)$, where:

- L is a finite set of labels $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$ used to represent control locations. Each statement in a program has a unique label. For example, ℓ_0 usually indicates the starting location.
- $X = \{x_0, \ldots, x_n\}$ is a set of program variables and $R = \{r_1, \ldots, r_m\}$ is a set of sampling variables.
- $\mapsto$ is a transition relation. Its element is in the form (ℓ, α, ℓ'), representing one step of transition from control location ℓ to ℓ', by an update function α: $\mathbb{R}^{|X \cup R|} \rightarrow \mathbb{R}^{|X|}$.
- $\perp$ is a sign of the exit of the program.

```
1: while (h≤t){
2:     p=Binomial(1,0.5);
3:     if (p>0)
4:         h=h+unifInt(0,10);
5:     t=t+1;
6: }
```

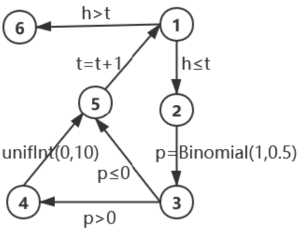


Figure 2: A probabilistic program with its CFG

Example 1. In Figure 2, we show a probabilistic program on the left and its control flow graph on the right. In the program, t is a program variable, p and h are smapling variables. Here, two probability distributions are used: **Binomial**(1,0.5) represents a binomial distribution, which yields values 0 and 1 with equal chance; **unifInt**(0,1) represents an integer uniform distribution, which yields the integers between 0 and 10 with equal probability. The numbers 1-6 on the left are the program control locations, where 1 is the starting location and 6 is the terminal location. The probabilistic program simulates a scene of the tortoise hare race. The tortoise's initial position is t and the hare's initial position is h. The tortoise always moves at the speed of 1. With probability 0.5, the hare increases its speed by a random number between 0 and 10. The program terminates when the hare passes the tortoise.

3 TERMINATION ANALYSIS

Termination analysis is an important part of program verification. If a loop program terminates for all initial values that can enter the loop, the program is said to be terminating. Ensuring termination is a necessary condition for many properties of programs such as total correctness. Generally speaking, the termination of a program is undecideable [16], but for some subclasses of programs, the termination can be verified. In this paper, we prove the termination of the affine probabilistic programs by synthesizing ranking supermartingales [4]. PROPER implements Algorithm 1 [5, 13] to analyze probabilistic termination qualitatively and quantitatively.

Algorithm 1 Termination Analysis.

```
Input: Program P;
Output: Judge whether the program is terminating, isT;
    The probability to terminate afer N steps decreases exponentially, N.
1: Set template $g(\ell, \boldsymbol{x})$ with a natural number as the maximal degree. Each location has the common template with unique coefficient. If $\ell_\perp$, the template $g(\ell, \boldsymbol{x})$=K.
2: Traverse the programs parse tree
    2.1: Collect the invariant I[] for each location.
    2.2: Calculate the pre-expectation for each location.
        case "assignment statement" :
            $pre(\ell, \boldsymbol{x}) = \mathbb{E}(g(\ell', \alpha, \boldsymbol{x})$.
        case "condition(if-else) or loop(while) statement" :
            $pre(\ell, \boldsymbol{x}) = g(\ell', \boldsymbol{x})$.
        case "terminal statement" :
            $pre(\ell, \boldsymbol{x}) = g(\ell, \boldsymbol{x}) = K$.
3: By the concept of half-space, $H = g(\ell, \boldsymbol{x}) - pre(\ell, \boldsymbol{x}) - \epsilon$, where $\epsilon > 0$ and $K \le -\epsilon$
4: Pattern extraction. By Farkas' Lemma, $H' = \sum_{i=1}^{d} a_i \mu_i$, where $\mu_i \in \mu$, $\mu = \{\prod_{i=1}^{n} I_i | n \in \mathbb{N}_0 and I_1, \ldots, I_n \in I\}$ and $a_i$ is a non-negative real number.
5: Solve linear programming. H and H' are corrsponding coefficient relations. If solvable, the program P can be terminating, otherwise return.
6: Calculate the upper bound of expected termination time according to $EP(P) \le UB(P) := \frac{g(\ell_0, x_0) - K}{\epsilon}$
7: Calculate the difference interval $[a, b]$ satisfying the condition $a \le g(\ell', \boldsymbol{x}) - g(\ell, f(\boldsymbol{x}, \boldsymbol{r})) \le b$
8: Obtain N, according to $\mathbb{P}(\boldsymbol{T_p} > N) \le e^{-\frac{2(\epsilon(N-1) - g(\ell_0, x_0))^2}{(N-1)(b-a)^2}}$
```

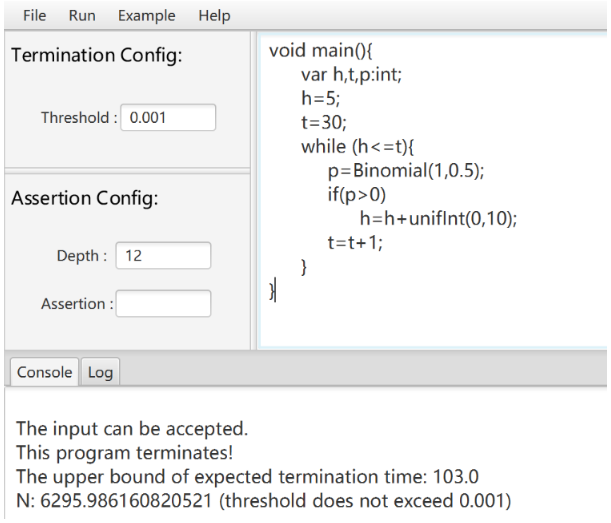
Qualitative analysis. It mainly analyzes whether a probabilistic program will terminate with probability 1. The specific idea is to calculate the martingale of each location and the value of location ℓ' minus the value of location ℓ is ϵ, where ϵ is greater than 0. Refer to Algorithm 1 steps 1 to 5. The polynomial ranking supermartingale (RSM) is a collection of functions at each label, represented by the template $g(\ell, \cdot)$. Each function is non-negative at non-terminal program position. So this method can not handle the distribution without lower bound, such as Gaussian distribution. The invariants and pre-expectation [4] are calculated by traversing the program parse tree. The pre-expectation is about the expression over the variables across a transition $\mapsto$. Then, through steps 3 and 4, H and H' can be computed. According to Farkas' Lemma [12], we know that the coefficients of H and H' match exactly. So we obtain a linear programming problem. For the sake of simplicity, in PROPER, we take the maximal degree of the template as 2, that is, the template $g(\ell, \cdot)$ is a quadratic equation. In addition, ϵ is taken as 1 and K as -1 for our tool PROPER. Below we show the RSM of Example 1 in Table 1.

Table 1: The RSM for Example 1

Label	Invariant	The RSM g
1	$h \leq t + 9$	$3 \cdot t - 3 \cdot h + 27$
2	$h \leq t$	$3 \cdot t - 3 \cdot h + 26$
3	$h \leq t$	$3 \cdot t - 3 \cdot h - 14 \cdot p + 32$
4	$h \leq t \wedge p = 1$	$3 \cdot t - 3 \cdot h + 17$
5	$h \leq t + 10$	$3 \cdot t - 3 \cdot h + 31$
6	$t < h$	-1

The "Label" column lists control locations of the program. The "Invariant" column gives the logical formulas that the reachable labelled locations satisfy when the program is executed from the starting location. The "The RSM g" column denotes the ranking supermartingale on the labels. The expected value of the RSM decreases by at least a positive amount ϵ after each execution of a statement. For example, when going from from label 1 to label 2, the expected value of the supermartingale decreases by 1. Similarly in the step from label 2 to label 3, the condition is met because $\mathbb{E}(Binomial(1, 0.5)) = 0.5$.

Quantitative analysis. We aim to estimate the upper bound of expected termination time and calculate the bound N, so that the probabilistic program concentrates on termination before N steps. We focus on the approximation of the expected termination time, as given in steps 6 to 8 in Algorithm 1. According to the previous steps, we can find the coefficients of the polynomial template RSM. The existence of an RSM ensures a finite upper bound on the expected termination time. If we know the initial values of variables, we can see that the value at the first location is $g(\ell, x_0)$, the value is K at the terminated location $\ell_\perp$ and the difference between two consecutive locations is ϵ. Therefore, when program P is almost surely terminating, we can get the upper bound on termination time for the given initial condition: $ET(P) \leq UB(P) = \frac{g(\ell, x_0) - K}{\epsilon}$. Then we focus on the concentration problem, that is, the probability of termination after N steps shows an exponential decrement. Exponential sum is one of the most commonly used specific function families in nonlinear approximation theory. Our main idea is based on martingale inequality of Azuma's Inequality [2], Hoeffding's Inequality [10, 14] and Bernstein's Inequality [3, 14]. In probability theory, the Azuma's inequality gives a concentration result for the values of martingales that have bounded differences. Hoeffding's Inequality is a special case of Azuma's Inequality. It proposes an upper bound on the probability that the sum of random variables deviates from its expected value. Bernstein's Inequality is a generalization of Hoeffding. It can handle not only independent variables but also weak independent variables. By Chatterjee *et al.* [13], we know that if $\epsilon(N - 1) > g(\ell_0, \boldsymbol{x_0})$, the inequality $\mathbb{P}(\boldsymbol{T_p} > N) \leq e^{-\frac{2((N-1)-g(\ell_0, x_0))^2}{(N-1)(b-a)^2}}$ holds. Consider again Example 1 with the initial values t=30 and h=5. We have $ET(P) \leq \frac{3 \cdot 30 - 3 \cdot 5 + 27 - K}{\epsilon} = 103$, where K and ϵ are set to be -1 and 1, respectively. The difference interval exists, which is [-28,14]. Suppose we make the query $\mathbb{P}(\boldsymbol{T_p} > N) \leq 10^{-3}$, the number N can be computed to be 6296 approximately.



Threshold : 0.001

Depth : 12

Assertion :

```
void main(){
    var h,t,p:int;
    h=5;
    t=30;
    while (h<=t){
        p=Binomial(1,0.5);
        if(p>0)
            h=h+unifInt(0,10);
        t=t+1;
    }
}
```

```
The input can be accepted.
This program terminates!
The upper bound of expected termination time: 103.0
N: 6295.986160820521 (threshold does not exceed 0.001)
```

Figure 3: PROPER User Interface

The user interface is shown in Figure 3. The concentration termination time can be set on the upper left of the interface and the probabilistic programs can be written on the right. Alternatively, a probabilistic program can be imported from an external file. Then by clicking “Termination Analysis" in the “Run" drop-down menu, we start the analysis process. The analysis results are displayed in the console.

4 ESTIMATING THE PROBABILITIES OF ASSERTIONS

The goal of this section is to estimate the probability that a given assertion is correct at the exit of a program. For the specific algorithm, we refer to Sankaranarayanan *et al.* [15]. There are two main steps.

The first step is to generate a sufficient and appropriate path set S with high confidence coverage. Using symbolic execution [8], the finite set $S = \{s_1, \ldots, s_i, \ldots, s_n\}$ contains distinct paths that are terminating. In other words, we have the transition sequences

Table 2: Termination analysis

Example	x_0	g(ℓ_0, x_0)	UB(P)	N (time≤0.01 sec)
Simple	$x = 100$	$6x$	601	1474.82
NestedLoop	$x = 1, m = 2$	$1040 \cdot m - 1040 \cdot x + 208$	1249	158141.27
Award	$bonus = 0$	$-4.0 \cdot bonus + 440$	441	4522.28
RandomWalk	$position = 0$	$-20 \cdot position + 120$	121	4695.66
Gambler	$money = 3$	$400 \cdot money + 400$	1601	1506477.30
Gambler2	$money = 10$	$45.454545 \cdot money + 454.545455$	910.09	1297006.47
Bitcoin mining	$coin = 10$	$5.317601 \cdot coin$	54.18	6.387839E7
Tortoise-Hare	$h = 5, t = 30$	$3 \cdot t - 3 \cdot h + 27$	103	4264.27

$s_i : \ell_0 \to \cdots \to \ell_k \to \cdots \to \ell_\perp$. The second step is to estimate the probability of a given assertion. For the path set S collected above, we can analyze each path in turn, then estimate the path probability and eventually the assertion probability. Since the calculation of the accurate probabilities is closely related to the volume of n-dimensional convex polyhedron, when the dimension increases, the calculation becomes very difficult and the time complexity is high [1]. Usually, the computation may fail due to the calculation or insufficient memory. On the premise of weighing the efficiency and accuracy of calculation, we focus on calculating the probability interval of a given assertion rather than the estimated value.

5 EXPERIMENTAL RESULTS

In this section, we present some experimental results of analyzing probabilistic programs using PROPER. More details can be found at GitHub.

We have written some simple but typical probabilistic programs with ‘while’ loops. In Table 2, we display the experimental results about termination analysis, where x_0 means the initial value of each variable, $g(\ell_0, x_0)$ is the polynomial ranking supermartingale about the starting location, $UB(P)$ is the upper bound of expected termination time and N is the bound that the probability of termination after N steps shows an exponential decrement (we set the threshold to be 0.01).

Table 3: Estimating the probability intervals of assertions

Ex.	Assertion	c	Bounds
framingham	$points \geq 10$	0.9636	[0.137343, 0.173695]
	$pointsErr - points \geq 5$	0.9365	[0.778149, 0.849318]
	$points - pointsErr \geq 5$	0.9326	[0.177619, 0.246763]
sum-three	$x > 5 \&\& y > 0$	0.9886	[0.150535, 0.163590]
	$x > 0 \&\& y > 0 \&\& z > 0$	0.9886	[0.125582, 0.137293]
	$x + y > z + 10$	0.9886	[0.447672, 0.470723]
	$x + y + z > 100$	0.9886	[0.012422, 0.025870]

The results of estimating the probability intervals of assertions as shown in Table 3 are consistent with those in [15]. A little difference lies in the syntax of assertions. We can support the conjunction operator “&&" in PROPER that are not available before. It can analyze situations where two or more conditions are met at the same time. In the table, an assertion is a boolean expression; c is the lower bound of path coverage; the column headed by Bounds gives the upper and lower bounds of the given assertion.

6 RELATED TOOLS

Sofware tools for the analysis of probabilistic programs has not yet received much attention. As far as we know, the only existing tools are ProbFuzz [6], PSense [11] and PSI [7]. PSI is a symbolic inference tool that approximates the probability density function represented by a probabilistic program. PSense is an automated verification tool that generates tight upper bounds for the sensitivity of probabilistic programs over initial inputs. ProbFuzz is a tool for testing probabilistic programs. Both the aforementioned tools do not consider termination analysis of probabilistic programs. For example, ProbFuzz focuses on testing rather than verification of probabilistic programs, PSI considers only inference and PSense solves sensitivity instead. Moreover, PSI/PSense requires that the input probabilistic while loop has a bounded number of loop iterations, while we can handle probabilistic loops with an unbounded number of loop iterations.

7 CONCLUSIONS AND FUTURE WORK

Uncertainty exists in many software systems, including data analysis, stochastic algorithms and Monte Carlo simulation [9]. We have designed PROPER to provide convenience and support for analyzing probabilistic programs. PROPER is helpful to perform qualitative and quantitative analysis on the termination of probabilistic programs. It can also collect path sets with high confidence coverage and compute probability intervals for assertions to hold in probabilistic programs.

However, we have only solved some aspects of the complex problem of probabilistic program analysis and verification. There are still many ways to improve the tool.

(1) Currently, PROPER only deals with linear programs. That is, it cannot handle variable multiplication, division and exponents, etc., regardless of the termination analysis or the estimation of assertion probability intervals.
(2) Non-deterministic probabilistic programs are also not supported. PROPER requires the the behavior of the input program to be fully probabilistic, and there is no non-deterministic transitions.
(3) In the future, we hope to find a better way to compute accurate termination time and probability intervals under the premise of ensuring high efficiency.

REFERENCES

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. 1998. Proof verification and the hardness of approximation problems. *Journal of the Acm* 45, 3 (1998), 501–555.

[2] K. Azuma. 1967. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal* 19, 3 (1967), 357–367.

[3] G. Bennett. 1962. Probability inequalities for the sum of independent random variables. *J. Amer. Statist. Assoc.* 57, 297 (1962), 33–45.

[4] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044*. Springer, Berlin, Heidelberg, 511–526.

[5] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs through Positivstellensatz's. *Computer Aided Verification* 9779 (July 2016), 489–501. DOI: http://dx.doi.org/10.1007/978-3-319-41528-4_1

[6] Saikat Dutta, Owolabi Legunsen, Zixin Huang, and Sasa Misailovic. 2018. Testing probabilistic programming systems. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 574–586. DOI: http://dx.doi.org/10.1145/3236024.3236057

[7] Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I (Lecture Notes in Computer Science)*, Swarat Chaudhuri and Azadeh Farzan (Eds.), Vol. 9779. Springer, 62–83. DOI: http://dx.doi.org/10.1007/978-3-319-41528-4_4

[8] J. Geldenhuys, M. B. Dwyer, and W. Visser. 2012. Probabilistic symbolic execution. In *ISSTA*. ACM, 166–167.

[9] HASTINGS and K. W. 1970. Monte Carlo sampling methods using Markov chains and their applications. 57, 1 (1970), 97–109.

[10] W. Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.* 58, 31 (1963), 13–30.

[11] Zixin Huang, Zhenbang Wang, and Sasa Misailovic. 2018. PSense: Automatic Sensitivity Analysis for Probabilistic Programs. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings (Lecture Notes in Computer Science)*, Shuvendu K. Lahiri and Chao Wang (Eds.), Vol. 11138. Springer, 387–403. DOI: http://dx.doi.org/10.1007/978-3-030-01090-4_23

[12] Farkas J. 1894. A fourier-féle mechanikai elv alkalmazásai (Hungarian). *Mathematikaiés Természettudományi Értesit´o* 12 (1894), 457–472.

[13] Chatterjee K, Fu H, and Novotny P. 2015. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Sigplan Notices* 51, 1 (June 2015), 327–342.

[14] C. McDiarmid. 1998. Concentration. In Probabilistic Methods for Algorithmic Discrete Mathematics. (1998).

[15] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static Analysis for Probabilistic Programs: Inferring Whole Program Properties from Finitely Many Paths. In *PLDI*. 447–458.

[16] Alan Turing. 1936. On Computable Numbers, with an Application to the Entscheidungs problem. *London Mathematical Society* 42, 2 (1936), 230–265.