Probabilistic Programs Analysis and Verification

Anonymous Author(s)

ABSTRACT

Probabilistic programs combine probabilistic reasoning model with Turing complete programming language, unify the formal description of calculation and uncertain knowledge, can effectively deal with complex relational models and uncertain problems. This paper presents PMars (Open Source Codes: https://github.com/), a verification tool based on probabilistic program, which includes the following two functions. On the one hand, it is to estimate the correct probability interval for given assertions, which helps to study the influence of uncertain variables on the results of probabilistic programs. On the other hand, it is to analyze the probabilistic programs' liveness property of termination qualitatively and quantitatively. Qualitative refers to whether the program terminates with probability 1. Quantitative includes estimating the expected termination time and calculating N, which the termination probability decreases exponentially after N steps. They can all effective execution in polynomial time. PMars supports basic probabilistic language syntax, with more than ten kinds of probability distribution functions built-in. We have verified through various programs that the experimental results show the effectiveness of PMars.

CCS CONCEPTS

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

KEYWORDS

Probabilistic Programming, Program Verification, Probability Interval, Termination

ACM Reference Format:
Anonymous Author(s). 1997. Probabilistic Programs Analysis and Verification. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97).* 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 requires us to use prerequisite knowledge and deductive reasoning to predict the results, that is to say, to make decisions on nondeterministic problems through probability reasoning. Therefore, probabilistic programs is produced. Probabilistic programs is a kind of logic program with probabilistic facts. It makes the probabilistic reasoning model easier to build and can estimate the possibility of the expected situation. Probabilistic programs has a wide and flexible scope of application. It is used 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 [5], etc. The analysis of programs with probability variables has also received widespread attention in academia and industry. Therefore, we need to learn the impact of probabilitisic variables on the program results, analyze and verify the properties of probabilistic grams, so as to ensure that the final results of the program have good reliability and stability.

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: Partial syntax specification of probabilistic programs language

This paper presents a verification tool PMars based on probabilistic programs, which can automatically analyze probabilistic language conforming to the specification (As shown in Figure **??**). It mainly includes two parts: one is to estimate the correct probability interval of given assertion when the program terminates; the other is to analyze the termination qualitatively and quantitatively.

More specifically, probabilistic programs validates tool PMars provides:

(1) Define and parse probabilistic programs language. More than ten kinds of probability distributions are built in, which is convenient to call for building probabilistic model.
(2) Reduce the estimation of given assertions to polyhedron solving problem. In addition, we support the probabilistic programs of infinite state.
(3) Reduce the termination analysis to linear programming problem. We also consider the concentration results under the premise that the program can be terminated.

2 PROBABILISTIC PROGRAMS

In order to analyze and verify probabilistic programs, we first need a probabilistic language with sufficient expressivity and easy to understand. We now consider the syntax [13] of probabilistic programs, whose syntax specifications are shown in Figure ??. The expressions in the probabilistic language is similar to the classic imperative programs, mainly composed of three statements: assignment statement, condition-branch statement (if-else) and loop statement (while). In addition, a variety of random value generators are built-in, as shown in Figure 1, which can be used to simulate different distribution situations. Variables are classified into two types: program variables and sampling variables. Program variables include integer variables, real variables and boolean variables. Boolean variables are mainly used for condition statement and loop statement. Sampling variables is assigned to when the program is running, which is subject to a continuous probability distribution or discrete probability distribution.

Table 1: Specification of the inbuilt random value generators

Name	Parameter	Density Function
R($\mathbb{X}$, $\mathbb{P}$)	$\sum_{n=1}^{N} p_n$=1	$P(X = x_k) = p_k$
Binomial(n,p)	0<p<1 n≥1	$\binom{k}{n} p^k (1-p)^{n-k}, k = 1, 2, \ldots$
Poisson(λ)	λ>0	$\frac{\lambda^k e^{-\lambda}}{k!}$
UnifInt(a,b)	a≤b	$\begin{cases} \frac{1}{b-a} & , b > a \\ 1 & , a = b \end{cases}$
UnifReal(a,b)	a<b	$\frac{1}{b-a}$
Exponential(θ)	θ>0	$\frac{1}{\theta} e^{\frac{x}{\theta}}$
Normal(μ,σ^2)	σ>0	$\frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$
Gamma(α,β)	α, β>0	$\frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{\frac{-x}{\beta}}, x > 0$
Beta(α,β)	α, β>0	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}$,0<x<1
Laplace(x|μ, λ)	$\lambda > 0$	$\frac{1}{2\lambda} e^{\frac{\lvert x-\mu \rvert}{\lambda}}$
Geometric(p)	0<p<1	$(1-p)^{k-1} p, k = 1, 2, \ldots$
T(n)	n≥1	$\frac{\Gamma(\frac{n+1}{2})}{\sqrt{n\pi}\Gamma(\frac{n}{2})} (1 + \frac{x^2}{n})^{-(n+1)/2}$

We use control flow graphs (CFGs) to express the semantics of probabilistic programs [12]. It is a tuple which takes the form (L,(X,R),↦,⊥), where:

- L is a finite set of labels L={ℓ_0,ℓ_1,...,ℓ_n} that represent the control locations. Each statement in a program has a unique label. Such as ℓ_0 indicates the starting location, ℓ_n indicates the last location.
- (X,R) is a set of variables, where X={x$_0$,...,x$_n$} is the set of program variables and R={r$_1$,...,r$_m$} is the set of sampling variables.
- ↦ is a transition relationship. ℓ_i is equivalent to a node in the control flow graph and ↦ is equivalent to transition from one node to another node. It can be expressed in the form of (ℓ,α,ℓ').
- ⊥ is a sign of the exit of the program.

3 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 the program. There are two main steps.

The First step is to generate sufficient and appropriate path set S with high confidence coverage. The finite set $S = \{s_1, \ldots, s_i, \ldots, s_n\}$ contains distinct paths that can be terminated, where $s_i : \ell_0 \to \cdots \to \ell_k \to \cdots \to \ell_\perp$. The control flow of loop statements on program variables and sampling variables may put the program in infinite state. Therefore, we adopt the method of simulating symbol execution. Execution paths can be represented as a tree. Through symbol execution [6], we traverse the program execution tree and collect the semantic state. So that each path of the program execution is based on the original probability distribution. In this way, our method is able to solve the problem even if program P contains program variables and sampling variables with a wide rang of distribution or even infinite state. We need to improve the coverage c of path set S to all paths in a finite time. Estimating the probability of a given assertion φ on path set S can be considered approximately as a whole program. We refer to some papers [14] that show the coverage c can be considered to be at least 95% with 99% confidence, when no new path is generated in 90 consecutive iterations.

The second step is to estimate the probability of a given assertion φ. For the path set S collected above, we can analyze each path in turn, then estimate the path probability and the assertion probability. Due to the calculation of accurate probability is closely related to volume of n-dimensional convex polyhedron. When the dimension increases, the calculation becomes more difficult and the time complexity is high [1]. Usually, it will 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 given assertion rather than estimated value.

Algorithm 1 mcBoundProbability

```
Input: Polytop p=(vars,constraints), maxDepth
Output: Probability interval[p1,p2]
 1: if isABox() || maxDepth<=0 then
 2:     [p1,p2]+=computeBoundingProbability();
 3: end if
    // If at least two (or more) non trivial clusters remain, then
    perform the decomposition.
 4: if isDecomposable(p) then
 5:     dim=selectBranchDimension();
 6:     branches=branchAndBound(dim,nBranch);
 7:     for b in branches do
 8:         mcBoundProbability(b,maxDepth-1);
 9:     end for
10: end if
```

Path probability. Each path has a series of variables and constraints. We need to estimate the probability of satisfying all constraints given the probability distributions on variables. Each variable can be regarded as a dimension of convex polyhedron. The probability problem can be reduced to polyhedron solving problem [15]. However, the solution of convex polyhedron by linear programming is too rough. Let us first decompose convex polyhedron into several non-intersecting convex bodies [7] and then add up the calculation results. As described in Algorithm 1, the isDecomposable() function is used to determine whether it can be decomposed. If true, it will be decomposed along the selected dimension. When the boundary box is found, the upper and lower bounds will be calculated by computeBoundingProbability() function. Based on the idea of importance sampling, selectBranchDimension() function selects a dimension which is unbounded or half bound or else select max width dimension. The branchAndBound() function decomposes according to the selected dimension, nBranch is the number of segmentation, the default is 2. In addition, to avoid infinite decomposition, we set maxDepth as the maximum recurrence depth, and our project is set as 12. Each path is calculated by the Algorithm 1. Because there is no intersection of each path, the path probability is equal to the sum of the each paths' probability. Let's assume the assertion probability is $[p_1, p_2]$. It indicates that the maximum 1-p_1 path has not been explored. This will be used to calculate the upper of assertion probability.

Assertion probability. The method of assertion probability is the same as path probability, also refer to Algorithm 1. The only difference is need to add assertion φ to conditional constraints. Let's assume the assertion probability is $[q_1, q_2]$. However, the path set S can't cover all paths in the program. We need to consider the probability of unexplored paths and add the probability beyond the path set S to the upper bound of assertion probability. Then the probability interval of given assertion is $[q_1, q_2 + (1 - p_1)]$.

4 TERMINATION ANALYSIS

Termination analysis [4] is an important part of program verification. Ensuring that the termination is a necessary condition for the correct verification of probabilistic program. In order to solve this problem, We have referred to some papers [3, 8, 11]. Decided to choose a powerful method to prove the termination of the probabilistic programs. That is, based on the concept of ranking supermartingale [2] analyze the probabilistic program qualitatively and quantitatively. Detailed as Algorithm 2

Qualitative analysis. It mainly analyzes whether the probabilistic programs will be terminated with probability 1 (almost sure termination) [10]. The specific idea is to calculate the supermartingale of each location and the value of location ℓ' does not exceed the location ℓ. Refer to Algorithm 2 step 1 to 6. Supermartingales are represented by template g with a natural number as the maximal degree. For the sake of simplicity, our tool takes 2, that is, the template g is a quadratic equation. Then calculate Pre-Expectation according to the formulas of step 3, where ℓ' is the next location of ℓ and f is the transition relation from location ℓ to ℓ'. We know that Pre-Expectation belong to convex set. The definition of half-space $\{x|\mathbf{a}^T\mathbf{x} \le b\}$ can be transformed into $\{x|(\mathbf{c}^\ell)^T\mathbf{x} \le d^\ell - \epsilon\}$ and c^ℓ, d^ℓ is a vector / scalar linear expression related to pre($\ell, \mathbf{x}$). we will apply it to Pre-Expectation, it is equivalent to $pre(\ell, \mathbf{x}) \le g(\ell, \mathbf{x}) - \epsilon$, and ϵ takes 1. Finally, according to Handelman's theorem, we can transform this problem into linear programming.

Algorithm 2 Termination Analysis.

```
Input: Program P;
Output: Judge whether the program is terminated, isT;
    The probability to terminate afer N steps decreases exponentially, N;
1: Set quadratic template $g(\ell, \mathbf{x})$. Each location has the common template with unique coefficient. If $\ell_\perp$, the template g=K.
2: Traverse the programs parse tree and collect the invariant I[] for each location.
3: Calculate the Pre-Expectation for each location.
       case "assignment statement" :
           $pre(\ell, \mathbf{x}) = \mathbb{E}(g(\ell', f(\mathbf{x}, \mathbf{r})), \mathbf{x})$.
       case "condition(if-else) or loop(while) statement" :
           $pre(\ell, \mathbf{x}) = g(\ell', \mathbf{x})$.
       case "terminal statement" :
           $pre(\ell, \mathbf{x}) = g(\ell, \mathbf{x})$.
4: According to the concept of half-space, $H = g(\ell, \mathbf{x}) - pre(\ell, \mathbf{x}) - \epsilon$
5: According to the concept of Handelman's Theorem, $H' = \sum_{i=1}^{d} a_i \cdot I_i$, where $\mu = \{\prod_{i=1}^{2} v\}$ $a_i$ is a non-negative real number.
6: Pattern extraction. The coefficient of H corresponds to that of H', which is converted into linear programming. If solvable, the program P can be terminated, otherwise return.
7: Calculate the difference-bounded [a,b] $(a \le g(\ell', \mathbf{x}) - g(\ell, f(\mathbf{x}, \mathbf{r})) \ge b)$
8: Obtain N, according to $\mathbb{P}(\boldsymbol{T_p} > N) \le e^{-\frac{2(\epsilon(N-1)-g(\ell_0, x_0))^2}{(N-1)(b-a)^2}}$
```

Quantitative analysis. Calculate the boundary N, so that the probability program concentrates on terminating before N steps. That is, the probability of termination after N steps shows an exponential decreases. We focus on the approximating of the expected termination time, refer to Algorithm 2 step 7 to 8. According to the previous steps, we can find the coefficients of polynomial template. If we know the initial value of variables, we can know that the first location is g(ℓ, x_0), the terminated location $\ell_\perp$ is K and the difference between each location is ϵ. For verification tool for probabilistic programs PMars, the value of K is -1 and the value of ϵ is 1. Therefore, when program P can be almost sure terminated, we can get the upper bound on termination time for the given initial condition is ET(P) $\le$ UB(P) = $\frac{g(\ell, x_0) - K}{\epsilon}$. 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, Hoeffding's Inequality and Bernstein's Inequality. In probability theory, the Azuma's inequality gives a concentration result for the values of martingales that have bounded differences. Hoeffding's Inequality [9] 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. Derivation from the paper [11], We know

Table 2: Experimental results:Termination Analysis

Example	x_0	g(ℓ_0, x_0)	UB(P)	N
Simple	$x = 100$	6x+2	603	2670
NestedLoop	$x = 1$	1040x+2290	1251	382807
Award	bonus=0	-4.0·bonus+442	443	10658
RandomWalk	$position = 0$	-20·position+122	123	11937
Gambler	$money = 3$	400·money+402	1603	3641131
Gambler2	$money = 10$	45.454545·money+456.545455	913	3134301
Bitcoin mining	$coin = 10$	5.317601·coin+2	57	1.532647E8
Infection	x=1000, p_1=0.25, p_2=0.1, q=0.3	21.107713+0.038215x+6.6$E - 5x^2$	59	215

when $\epsilon(N-1) > g(\ell_0, \mathbf{x_0})$, then $\mathbb{P}(T_{\mathbf{p}} > N) \le e^{-\frac{2((N-1)-g(\ell_0,x_0))^2}{(N-1)(b-a)^2}}$. Given the exponent, we can get the result of N.

5 EXPERIMENTAL RESULTS

In this section, we present our experimental results. For all the examples, you can view it on GitHub.

Table 3 presents the experimental results about estimating the probabilities bounds of assertions, where 'Assertion' is a Boolean expression, it can contain addition and subtraction operations and it also support the use of logical operator &&, 'c' is the lower bound of path coverage, 'Bounds' is the upper and lower bounds of the given assertion. Due to layout restrictions, some values are not presented in the table, such as the total number of execution paths and unique paths.

Table 3: Experimental results: Estimating the probabilities bounds of assertions

Ex.	Assertion	c	Bounds
carton	count≥5	0.9485	[0.948540,1]
	count≥10	0.9539,	[0.000639,0.046711]
	count≤7	0.9549	[0.918762,0.963832]
	totalWeight≥5.5	0.9386	[0.382145,0.453010]
	totalWeight≥6	0.9428	[0.246186,0.342997]
herman	count≥1	0.9561	[0.581128,0.625000]
	count≥20	0.9701	[0.000000,0.029876]
framin gham	points≥10	0.9636	[0.137343,0.173695]
	pointsErr-points≥5	0.9365	[0.778149,0.849318]
	points-pointsErr≥5	0.9326	[0.177619,0.246763]
sum-three	x+y>z+10	0.9886	[0.447672,0.470723]
	x+y+z>8	0.9886	[0.454482,0.475890]
	x+y+z>100	0.9886	[0.012422,0.025870]
	x>5 && y>0	0.9886	[0.150535,0.163590]
	x>0 && y>0 && z>0	0.9886	[0.125582,0.137293]
ckd-epi	f -f1≥0.1	0.9252	[0.351632,0.426397]
	f1 -f≥0.1	0.9261	[0.387275,0.461157]

We have written some simple but classical probabilistic programs with 'while' loop-statements. In Table 2, we present the experimental results about termination analysis, where 'x_0' means the initial value of variable, 'g(ℓ_0, x_0)' is the polynomial ranking supermartingale about the starting location, 'ET(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 decreases (we set the exponent to 1.5 $\cdot 10^{-5}$).

6 CONCLUSIONS AND FUTURE WORK

At present, uncertainty exists in many software systems, including data analysis, stochastic algorithm and Monte Carlo simulation. The design of PMars tool is to provide convenience and method for inference problem of probabilistic program system. PMars defines the basic syntax with probability distribution, which is used to model of probabilistic programs. PMars can collect path set S with high confidence coverage and calculate probability interval and assertion probability interval of each path. In addition, based on the idea of ranking-supermartingales, PMars can also perform qualitative and quantitative analysis on the termination of probabilistic programs.

However, we have only solved some aspects of the complex problem of probabilistic programs analysis and verification. We still have some areas for improvement:

(1) Limited to linear program. PMar can't deal with the cases of variable multiplication, division and exponents, etc., regardless of the estimation of the assertion probability interval or the termination analysis,
(2) Non-deterministic probabilistic programs is not supported. PMars can handle random variables with know probability distribution and probabilistic transitions. But for programs with uncertain variables and non-deterministic transitions, it can't handle them.
(3) We hope to find a better way to get more accurate probability interval under the premise of ensuring 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] 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.

[3] 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

[4] Michael Codish and Cohavit Taboch. 1999. Semantic basis for the termination analysis of logic programs. *Journal of Logic Programming* 41, 1 (1999), 103–123.

[5] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Databases* 9 (January 2014), 211–407. DOI:http://dx.doi.org/10.1561/0400000042

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

[7] Bingsheng He, Feng Ma, and Xiaoming Yuan. 2020. Optimally linearizing the alternating direction method of multipliers for convex programming. *Computational Optimization and Applications* 75 (March 2020), 361–388.

[8] Hermanns, Holger, Fioriti, Luis, Maria, and Ferrer. 2015. Probabilistic termination: soundness, completeness, and compositionality. In *POPL*. ACM, 489–501.

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

[10] Joe Hurd. 2002. A Formal Approach to Probabilistic Termination. In *Theorem Proving in Higher Order Logics, International Conference, Tphols, Hampton, Va, Usa, August*.

[11] 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.

[12] Dexter Kozen. 1981. Semantics of probabilistic programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350.

[13] Thomas Minka and John Winn. 2012. Compiler for probabilistic programs. (January 2012). DOI:http://dx.doi.org/US8103598B2

[14] 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.

[15] J. Geldenhuys W. Visser and M. B. Dwyer. 2012. Green: reducing, reusing and recycling constraints in program analysis. In *SIGSOFT FSE*. ACM, 1–11.