# TAP: 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 TAP, a tool for analyzing termination and assertions for probabilistic programs. On one hand, it can help to analyze the termination property of probabilistic programs both qualitatively and quantitatively. It can check whether a probabilistic program terminates with probability 1, estimate the 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 TAP is demonstrated through various probabilsitic programs.

## CCS CONCEPTS

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

## KEYWORDS

Probabilistic Programming, Program Verification, Probability Intervals, Termination

**ACM Reference Format:**
Anonymous Author(s). 2020. TAP: 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, 7 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 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 for 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 [6], etc. The analysis of probabilistic programs has also received widespread attention in academia and industry.

In the current work we present presents TAP, a verification tool for analyzing Termination and Assertions for Probabilistic programs. It can automatically analyze whether a probabilistic program conforms to a specification. It mainly includes two parts: one is to analyze the termination property of a given probabilistic program qualitatively and quantitatively; the other is to estimate the correct probability interval for a given assertion to hold. More specifically, TAP 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. Note that we support 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/.

## 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 [16] 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 as listed in Table 1, 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 is running, which is subject to a continuous or discrete probability distribution.


*ESEC/FSE 2020, 8 - 13 November, 2020, Sacramento, California, United States*

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**

**Table 1: Specification of the inbuilt random value generators**

| Name | Parameter | Density Function |
|---|---|---|
| R(𝕏, ℙ) | $\sum_{n=1}^{N} p_n$=1 | $P(X = x_k) = p_k$ |
| Binomial(n,p) | 0¡p¡1<br>n≥1 | $\binom{k}{n}p^k(1-p)^{n-k}, k = 1, 2, \ldots$ |
| Poisson($\lambda$) | $\lambda$¿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($\theta$) | $\theta$¿0 | $\frac{1}{\theta}e^{\frac{x}{\theta}}$ |
| Normal($\mu$,$\sigma^2$) | $\sigma$¿0 | $\frac{1}{\sqrt{2\pi}\sigma}e^{-(x-\mu)^2/(2\sigma^2)}$ |
| Gamma($\alpha$,$\beta$) | $\alpha, \beta$¿0 | $\frac{1}{\beta^\alpha\Gamma(\alpha)}x^{\alpha-1}e^{\frac{-x}{\beta}}, x > 0$ |
| Beta($\alpha$,$\beta$) | $\alpha, \beta$¿0 | $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$,0¡x¡1 |
| Laplace(x—$\mu, \lambda$) | $\lambda > 0$ | $\frac{1}{2\lambda}e^{\frac{\lvert x-\mu\rvert}{\lambda}}$ |
| 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 [15]. Formally, a CFG is a tuple in the form $(L, X, R, \mapsto, \bot)$, 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, $\ell_0$ 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 $(\ell, \alpha, \ell')$, representing one step of transition from control location $\ell$ to $\ell'$, by performing action $\alpha$.
- $\bot$ is a sign of the exit of the program.

## 3 TERMINATION ANALYSIS

Termination analysis [4] is an important part of program verification. Ensuring termination is a necessary condition for many properties of programs such as total correctness. A powerful technique for proving probabilistic termination is to synthesize ranking supermartingales [2, 3, 10, 14]. TAP implements Algorithm 1 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 quadratic template $g(\ell, \boldsymbol{x})$. Each location has the common template with unique coefficient. If $\ell_\bot$, 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" :
$pre(\ell, \boldsymbol{x}) = \mathbb{E}(g(\ell', f(\boldsymbol{x}, \boldsymbol{r})), \boldsymbol{x})$.
case "condition(if-else) or loop(while) statement" :
$pre(\ell, \boldsymbol{x}) = g(\ell', \boldsymbol{x})$.
case "terminal statement" :
$pre(\ell, \boldsymbol{x}) = g(\ell, \boldsymbol{x})$.
4: By the concept of half-space, $H = g(\ell, \boldsymbol{x}) - pre(\ell, \boldsymbol{x}) - \epsilon$
5: By Handelman's Theorem, $H' = \sum_{i=1}^{d} a_i \cdot I_i$, where $\mu = \{\prod_{i=1}^{2} \nu\}$ and $a_i$ is a non-negative real number.
6: Pattern extraction. The coefficient of $H$ corresponds to that of $H'$, which is converted into a linear programming problem. If solvable, the program P can be terminating, otherwise return.
7: Calculate the difference-bounded $[a, b]$ with $(a \leq g(\ell', \boldsymbol{x}) - g(\ell, f(\boldsymbol{x}, \boldsymbol{r})) \leq b)$
8: Obtain N, according to $\mathbb{P}(\boldsymbol{T_p} > N) \leq e^{-\frac{2(\epsilon(N-1)-g(\ell_0,x_0))^2}{(N-1)(b-a)^2}}$

*Qualitative analysis.* It mainly analyzes whether the probabilistic programs will terminate with probability 1 (almost sure termination) [13]. The specific idea is to calculate the supermartingale of each location and the value of location $\ell$' does not exceed that of location $\ell$. Refer to Algorithm 1 steps 1 to 6. Supermartingales are represented by the template $g$ with a natural number as its maximal degree. For the sake of simplicity, our tool takes 2, that is, the template $g$ is a quadratic equation. We calculate the pre-expectation for each location according to the formulas of step 3, where

$\ell$' is the next location of $\ell$ and $f$ is the transition relation from location $\ell$ to $\ell$'. Since the pre-expectations belong to a convex set. The definition of half-space $\{x|\mathbf{a}^T\mathbf{x} \leq b\}$ can be transformed into $\{x|(\mathbf{c}^\ell)^T\mathbf{x} \leq d^\ell - \epsilon\}$ and $c^\ell$ (resp. $d^\ell$ is a vector (resp. scalar) linear expression related to $pre(\ell, \boldsymbol{x})$. we will apply it to the pre-expectation, it is equivalent to $pre(\ell, \boldsymbol{x}) \leq g(\ell, \boldsymbol{x}) - \epsilon$, and $\epsilon$ takes 1. Finally, according to Handelman's theorem, we can transform this problem into a linear programming problem.

*Quantitative analysis.* We aim to calculate the boundary $N$, so that the probabilistic program concentrates on termination before $N$ steps. That is, the probability of termination after $N$ steps shows an exponential decrement. We focus on the approximation of the expected termination time, cf. steps 7 to 8 in Algorithm 1. According to the previous steps, we can find the coefficients of the polynomial template. 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 $\epsilon$. For our tool TAP, the value of $K$ is set to be $-1$ and the value of $\epsilon$ is 1. Therefore, when program $P$ is almost sure terminating, we can get the upper bound on termination time for the given initial condition is: $ET(P) \leq UB(P) = \frac{g(\ell, x_0) - K}{\epsilon}$. 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 [11] 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 [14], we know that when $\epsilon(N-1) > g(\ell_0, \boldsymbol{x_0})$, 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. Given the exponent, we can obtain the result of $N$.

## 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 the program. There are two main steps.

The first step is to generate a 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 are terminating, that is, we have the transition sequences $s_i : \ell_0 \to \cdots \to \ell_k \to \cdots \to \ell_\perp$. The control flow of loop statements on program variables and sampling variables may lead to infinitely many states. Therefore, we adopt the method of simulating symbolic executions. Execution paths can be represented using a tree. Through a symbolic execution [8], we traverse the program execution tree and collect the semantic states, so that each path of the program execution is based on the original probability distribution. In this way, our method is able to handle the problem that program $P$ contains program variables and sampling variables with a wide range of distributions or even infinitely many states. Then we need to improve the coverage $c$ of path set $S$ to all paths in a finite amount of time. Estimating the probability of a given assertion $\varphi$ on path set $S$ can be considered approximately as on the whole program. Our implementation is based on [17] where it is shown that 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 $\varphi$. 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.

**Algorithm 2** mcBoundProbability

**Input:** Polytop p=(vars, constraints), maxDepth
**Output:** Probability interval $[p_1, p_2]$

```
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 calculation of path probability can be reduced to a polyhedron solving problem [18]. However, the solution of a convex polyhedron by linear programming is too rough. Therefore, we first decompose the convex polyhedron into several non-intersecting convex bodies [9] and then add up the calculation results. As described in Algorithm 2, the function *isDecomposable*() 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 the function *computeBoundingProbability*(). Based on the idea of importance sampling, the function *selectBranchDimension*()

selects a dimension which is unbounded or half bound or else it selects the maximum width dimension. The function *branchAndBound*() performs the decomposition according to the selected dimension. Here, *nBranch* is the number of segmentation, with the default value 2. In addition, to avoid infinite decomposition, we set *maxDepth* as the maximum recurrence depth, with the default value 12 in TAP. Each path is calculated by the Algorithm 2. Since there is no intersection of each path, the path probability is equal to the sum of each paths' probability. Let us assume that 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 bound of the assertion probability.

*Assertion probability.* The method of computing an assertion probability is similar to that of computing a path probability. The only difference is the need to add assertion $\varphi$ to the conditional constraints. Let us assume that the assertion probability is $[q_1, q_2]$. Since the path set $S$ may not cover all the 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 the assertion probability. Then the probability interval of the given assertion is $[q_1, q_2 + (1 - p_1)]$.

## 5 EXPERIMENTAL RESULTS

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

We have written some simple but classical 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 exponent to $1.5 \cdot 10^{-5}$).

Table 3 shows the experimental results about estimating the probability interval of assertions. In the table, an assertion is a boolean expression that can contain addition and subtraction operations and also supports the use of the logical operator conjunction; $c$ is the lower bound of path coverage; the column headed by Bounds gives the upper and lower bounds of the given assertion.

**Table 3: Experimental results: estimating the probability interval 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] |
| framingham | 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 | $f - f_1 \geq 0.1$ | 0.9252 | [0.351632, 0.426397] |
| | $f_1 - f \geq 0.1$ | 0.9261 | [0.387275, 0.461157] |

**Table 2: Experimental results: termination analysis**

| Example | $x_0$ | g($\ell_0, x_0$) | 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, $q = 0.3$ | $21.107713 + 0.038215x + 6.6E - 5x^2$ | 59 | 215 |

## 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 [5], PSense [12] 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 generate 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. We have designed TAP to provide convenience and support for analyzing probabilistic programs. TAP 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 interval 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, TAP 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 interval.
(2) Non-deterministic probabilistic programs are also not supported. TAP 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 more accurate probability interval 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] 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] 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

[6] 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

[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] 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.

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

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

[12] 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

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

[14] 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.

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

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

[17] 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.

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