SuperDP: Differential Privacy Refutation via Supermartingales

KRISHNENDU CHATTERJEE, IST Austria, Austria
EHSAN KAFSHDAR GOHARSHADY, IST Austria, Austria
ĐORĐE ŽIKELIĆ, Singapore Management University, Singapore

Differential privacy (DP) has established itself as one of the standards for ensuring privacy of individual data. However, reasoning about DP is a challenging and error-prone task, hence methods for formal verification and refutation of DP properties have received significant interest in recent years. In this work, we present a novel method for automated formal refutation of ϵ-DP. Our method refutes ϵ-DP by searching for a pair of inputs together with a non-negative function over outputs whose expected value on these two inputs differs by a significant amount. The two inputs and the non-negative function over outputs are computed simultaneously, by utilizing upper expectation supermartingales and lower expectation submartingales from probabilistic program analysis, which we leverage to introduce a sound and complete proof rule for ϵ-DP refutation. To the best of our knowledge, our method is the first method for ϵ-DP refutation to offer the following four desirable features: (1) it is fully automated, (2) it is applicable to stochastic mechanisms with sampling instructions from both discrete and continuous distributions, (3) it provides soundness guarantees, and (4) it provides semi-completeness guarantees. Our experiments show that our prototype tool SuperDP achieves superior performance compared to the state of the art and manages to refute ϵ-DP for a number of challenging examples collected from the literature, including ones that were out of the reach of prior methods.

CCS Concepts: • **Software and its engineering** → **Automated static analysis**; **Software verification**; **Automated static analysis**; **Software verification**; • **Mathematics of computing** → *Probability and statistics.*

Additional Key Words and Phrases: Static Program Analysis, Differential Privacy, Probabilistic Programming, Martingales

ACM Reference Format:
Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, and Đorđe Žikelić. 2026. SuperDP: Differential Privacy Refutation via Supermartingales. *Proc. ACM Program. Lang.* 10, PLDI, Article 218 (June 2026), 25 pages. https://doi.org/10.1145/3808296

1 Introduction

Differential privacy. Differential privacy is a formal notion of privacy that tackles the problem of extracting statistical information about a dataset, without compromising the privacy of individual data subjects [31, 33]. It has gained great prominence over the recent years as one of the standards for ensuring privacy in (randomized) computation [11, 29, 41]. Intuitively, a randomized algorithm or a stochastic mechanism is said to be differentially private, if slightly changing the input of the algorithm (e.g. changing the data of a single user) does not significantly affect the probability distribution of outputs. Formally, given a probabilistic program P modeling the randomized

Authors' Contact Information: Krishnendu Chatterjee, IST Austria, Klosterneuburg, Austria, krishnendu.chatterjee@ist.ac.at; Ehsan Kafshdar Goharshady, IST Austria, Klosterneuburg, Austria, ehsan.goharshady@ist.ac.at; Đorđe Žikelić, Singapore Management University, Singapore, Singapore, dzikelic@smu.edu.sg.

ACM 2475-1421/2026/6-ART218
https://doi.org/10.1145/3808296

computation, a similarity relation $\sim_\Phi$ over its inputs, and a privacy budget $\epsilon \geq 0$, P is said to be *ϵ-differentially private (ϵ-DP)* if for every pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ the resulting output distributions of P are not too distant from each other, i.e. the inequality

$$\mathbb{P}_{\mathbf{x}^1_{in}}\Big[Output(A)\Big] \leq e^\epsilon \cdot \mathbb{P}_{\mathbf{x}^2_{in}}\Big[Output(A)\Big], \tag{1}$$

holds for every set of outputs A in P. Smaller values of ϵ provide stronger privacy guarantees, with $\epsilon = 0$ ensuring perfect privacy and the indistinguishability of computation due to similar inputs.

Correctness of differential privacy. While DP provides an intuitive and mathematically clean definition of privacy, it turns out that reasoning about it is a difficult task. For instance, minor tweaks to correct mechanisms can break differential privacy [32, 43]. Moreover, as any non-trivial property of (probabilistic) programs, the problem of DP checking for Turing-complete probabilistic programming languages is undecidable [6]. This inherent difficulty, together with the importance and wide adoption of differential privacy as one of the standards for private computation, lead to a significant interest and research effort towards enabling automated reasoning about DP.

Automated reasoning about DP can be divided into two complementary problems – (1) *DP verification*, i.e. the problem of proving that a stochastic mechanism is differentially private, and (2) *DP refutation*, i.e. the problem of proving that a stochastic mechanism is *not* differentially private. Due to the undecidability of DP checking, one cannot hope for a single algorithm that can verify and refute DP for all stochastic mechanisms, hence these two problems require different approaches.

The DP verification problem has received significant attention, see e.g. [5–8, 12, 59], and we discuss existing methods and approaches in Section 7. The DP refutation problem, on the other hand, has received comparatively less attention. Existing automated methods for DP refutation can be classified into static and dynamic methods. Dynamic methods aim to refute DP by sampling program executions and deriving statistical lower bounds on distances between output distributions on different inputs. If these distances are sufficiently significant, one can conclude that the mechanism is not DP with statistical guarantees [30]. Static methods, on the other hand, aim for formal guarantees by performing static analysis of probabilistic programs that implement the stochastic mechanism of interest. However, most existing static analyses for DP refutation are either not fully automated, or are restricted to finite or countable programs that do not allow real-valued program variables or sampling from continuous distributions [6, 34], which is a significant limitation given that continuous distributions such as Laplacians are standard in many DP mechanisms used in practice [31, 33, 54]. A notable exception is CheckDP [59] (successor of LigthDP [63] and ShadowDP [60]), which searches for two program inputs whose respective executions cannot be aligned to yield the same output without changing the program by much. They then use the off-the-shelf exact inference tool PSI [36] to formally prove that ϵ-DP is violated for the identified pair of inputs.

Limitations of previous approaches. While the above works present significant advances in automated reasoning about DP, the existing support for automated and formal DP refutation is still limited and DP refutation remains a challenging problem. In particular, to the best of our knowledge, no existing method for DP refutation provides all of the following desirable features:

(1) *Automation.* We are interested in fully automated methods for DP refutation.
(2) *Support for general probabilistic programs.* We are interested in methods that are applicable to stochastic mechanisms that allow sampling from both discrete and continuous distributions. As discussed above, this is highly important because continuous distributions, such as Laplacians, are standard in many classical DP mechanisms [31, 33, 54].
(3) *Soundness guarantees.* We are interested in static analysis methods that can provide formal guarantees on the correctness of DP refutations that they report.

(4) *Semi-completeness guarantees.* Given the undecidable nature of DP checking, one cannot hope for an algorithm which is sound and complete. However, as is the case in many static program analyses such as termination [27, 50, 51] or safety [14, 26] proving, it can be beneficial to provide semi-completeness guarantees, i.e. a set of conditions under which a method is guaranteed to refute DP. This is particularly important in the context of DP analysis, given how sensitive DP mechansisms are to even minor tweaks [32, 43]. CheckDP [59], which is the only method that satisfies the above three features, does not provide any form of semi-completeness guarantees.

Our contributions. In this work, we propose a method for ϵ-DP refutation that provides all four of the desirable features discussed above. The key idea behind our method is to search for a pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ for which ϵ-DP is violated by showing that they lead to "expectation mismatch". That is, rather than explicitly searching for an output event A for which eq. (1) is violated, our method instead searches for a non-negative function f over the outputs of the stochastic mechanism for which

$$\mathbb{E}_{\mathbf{x}^1_{in}}[f] > e^\epsilon \cdot \mathbb{E}_{\mathbf{x}^2_{in}}[f]. \tag{2}$$

We prove that this is both a necessary and sufficient condition for the pair of inputs to violate ϵ-DP.

A challenge in computing such a function f is that the exact computation of the expected value of f on output is in general infeasible, as it may not always admit a closed-form expression [42]. To overcome this challenge, our method instead computes a pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ and a function f *simultaneously and together with* a lower bound $L \leq \mathbb{E}_{\mathbf{x}^1_{in}}[f]$ and an upper bound $U \geq \mathbb{E}_{\mathbf{x}^2_{in}}[f]$ on the two expected values, while in addition enforcing $L > e^\epsilon \cdot U$. These three inequalities together entail the inequality in eq. (2) and hence refute ϵ-DP.

To reason about and automatically compute upper and lower bounds on the expected value of f on output, we leverage the notions of upper expectation supermartingales (UESMs) and lower expectation submartingales (LESMs), which were shown to provide sound and complete proof rules for computing upper and lower bounds on the expected value of a function on output [19, 20]. This allows us to formulate what we call *expectation super/sub-martingale witness for ϵ-DP refutation*, which are tuples $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, L_f, U_f)$ that consist of a pair of similar inputs, a non-negative function f over outputs, and a U/LESM pair which give rise to expectation bounds that together imply eq. (2). We show that our expectation super/sub-martingale witness for ϵ-DP refutation provide a *sound and complete proof rule* for ϵ-DP refutation in stochastic mechanisms that can be modeled as probabilistic programs that allow general Borel-measurable arithmetic expressions and sampling instructions from both discrete and continuous probability distributions. We then use this sound and complete proof rule to design a *sound and conditionally semi-complete algorithm* for ϵ-DP refutation. Our algorithm is applicable to stochastic mechanisms that can be modeled as polynomial arithmetic probabilistic programs (while allowing sampling instructions from distributions whose probability density function is not polynomial, e.g. Laplacian or normal distributions), and it follows a template-based synthesis approach to compute an instance of an expectation super/sub-martingale witness for ϵ-DP refutation which can be specified in the real-valued polynomial arithmetic. Finally, we develop and experimentally evaluate a prototype tool SuperDP (**Super**martingale-based **D**ifferential **P**rivacy refutation), and our experiments demonstrate our method's practical applicability and favourable performance compared to the state of the art on automated ϵ-DP refutation.

Our contributions can be summarized as follows:

(1) *Theory: Sound and complete proof rule for ϵ-DP refutation.* We introduce *expectation super/sub-martingale witness for ϵ-DP refutation*, a sound and complete proof rule for ϵ-DP refutation in stochastic mechanisms that can be modeled as probabilistic programs that allow general

```
Input: x ∈ {0,1} ∧ out = 0
Sim: (out¹ = out² = 0)
l1: if prob(0.5) then
l2:   out := x
    else
l3:   if prob(0.5) then
l4:     out := 0
      else
l5:     out := 1
lt:
Output: out
```

(a) Randomized response mechanism (RR-1) [33]: provides plausible deniability by ensuring $(\ln 3)$-DP.

```
Input: q ∈ ℝ^n
Sim: (η¹ = η² = 0) ∧ (∀i. |q_i¹ − q_i²| ≤ 1)
       ∧ (∀i. q_i¹ ≠ q_i² ⇒ ∀j ≠ i. q_j¹ = q_j²)
l1: for i ∈ [0,n):
l2:   η := laplace(1)
l3:   q_i := q_i + η
lt:
```

```
l1: η := laplace(1)
l2: q_0 := q_0 + η
lt:
Output: q
```

(b) Histogram mechanism [33] (top) and its unrolling for $n = 1$ (bottom): given true query answers q, adds independent Laplacian perturbation to each element to satisfy 1-DP.

Fig. 1. Our two running examples for illustrating our approach to refuting ϵ-DP. In both cases, we use Input to denote the predicate that defines possible program inputs, Sim to denote the similarity relation over inputs, and Out to denote output program variables.

Borel-measurable arithmetic expressions and sampling instructions from both discrete and continuous probability distributions.

(2) *Automation: Sound and conditionally semi-complete algorithm for ϵ-DP refutation.* We design a sound and conditionally semi-complete algorithm for ϵ-DP refutation in stochastic mechanisms that can be modeled as polynomial arithmetic probabilistic programs.

(3) *Experimental evaluation.* Our experiments show that our sound and conditionally semi-complete algorithm is able to refute ϵ-DP in a number of challenging examples collected form the literature, including examples that no prior automated method could handle.

2 Overview

To illustrate the key ideas behind our method for refuting DP, we will consider the randomized response mechanism in Fig. 1a and the histogram mechanism in Fig. 1b, both of which are basic and fundamental mechanisms in the DP literature [33]. In both cases, we use probabilistic programs (PPs) as a language for specifying stochastic mechanisms whose DP we wish to analyze. PPs are classical imperative or functional programs extended with the ability to sample values from probability distributions and to assign sampled values to program variables, and they provide a general expressive framework for modeling and specifying stochastic models and protocols [38, 55].

Example 1 (Running example: Randomized response mechanism). *The PP in Fig. 1a implements the well-known "randomized response" mechanism [33]. This mechanism allows individuals to respond to sensitive queries while providing them with plausible deniability. Suppose that input x is sensitive information, e.g. whether or not a surveyed student has cheated in an exam. The student is asked to toss a fair coin (line l_1), answer truthfully if it lands on heads (line l_2), and otherwise answer uniformly at random by throwing another fair coin (lines l_3 to l_5). This way, even if the student responds 'yes', they can credibly deny their cheating by claiming that the first coin landed on tails.*

Table 1. Output distribution of the randomized response mechanism in Fig. 1a.

in \ *out*	0	1
$(0,0)$	0.75	0.25
$(1,0)$	0.25	0.75

Example 2 (Running example: Histogram mechanism). *The PP in Fig. 1b implements the histogram mechanism introduced in [33]. The input to this mechanism is a vector of true answers to some number of queries q about a database (e.g. a query q_i may ask for the number of database elements that are equal to some given value). The goal of the histogram mechanism is to output the query answers privately by adding noise to them, where the noise is sampled according to the Laplacian distribution with mean 0 and scale $\frac{1}{\epsilon}$. Here, we consider $\epsilon = 1$ for illustration.*

Differential privacy refutation. Let P be a PP that models a stochastic mechanism whose differential privacy we wish to analyze. Intuitively, we say that P is DP, if for any pair of inputs to P that are sufficiently "similar", the probability of every output event on these two inputs is also similar.

This intuition is formalized as follows. Let $\mathcal{V}$ be the set of program variables in P with the set of output variables $\mathcal{V}_{out} \subseteq \mathcal{V}$, $\theta_{in} \subseteq \mathbb{R}^{|\mathcal{V}|}$ be the set of all inputs to P, and $\Phi \subseteq \theta_{in} \times \theta_{in}$ be a similarity relation over the inputs that specifies which input pairs are deemed to be "similar". For a pair of inputs $\mathbf{x}^1_{in}, \mathbf{x}^2_{in} \in \theta_{in}$, we write $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ to denote that $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}) \in \Phi$. Then, for $\epsilon \geq 0$, we say that the PP P is *ϵ-differentially private (ϵ-DP)*, if for any pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ and for any event A over the PP outputs, we have

$$\mathbb{P}_{\mathbf{x}^1_{in}}\Big[Output(A)\Big] \leq e^\epsilon \cdot \mathbb{P}_{\mathbf{x}^2_{in}}\Big[Output(A)\Big],$$

where we use $\mathbb{P}_{\mathbf{x}^1_{in}}$ and $\mathbb{P}_{\mathbf{x}^2_{in}}$ to denote the probability measures induced by the PP on these two inputs. In words, the probability of any output event on two similar inputs may differ by a multiplicative factor of at most e^ϵ. Given a PP P, a similarity relation Φ over its inputs and a privacy budget $\epsilon \geq 0$, the DP refutation problem is concerned with formally proving that P is not ϵ-DP.

Example 3. *The randomized response mechanism in Fig. 1a is ϵ-DP for $\epsilon = \ln 3$, but not ϵ-DP for $\epsilon < \ln 3$ [33]. Indeed, Table ?? shows the output distribution of the randomized response mechanism for each possible input pair (x, out). Based on the data in the table, it can be verified by inspection that $\mathbb{P}_{\mathbf{x}^1_{in}}[Output(A)] \leq e^{\ln 3} \cdot \mathbb{P}_{\mathbf{x}^2_{in}}[Output(A)]$ holds for all similar inputs $\mathbf{x}^1_{in} = (x^1, out^1), \mathbf{x}^2_{in} = (x^2, out^2) \in \{0,1\}^2$ and for all output events A. Hence, the mechanism is $(\ln 3)$-DP. However, the mechanism is* not *ϵ-DP for $\epsilon < \ln 3$. This is because, for the pair of similar inputs $(x^1, out^1) = (1,0)$ and $(x^2, out^2) = (0,0)$ and the output event $A = (out = 1)$, we have $\mathbb{P}_{\mathbf{x}^1_{in}}[Output(A)] = e^{\ln 3} \cdot \mathbb{P}_{\mathbf{x}^2_{in}}[Output(A)]$ so the inequality is tight for $\epsilon = \ln 3$ and would be violated for $\epsilon < \ln 3$.*

On the other hand, it is a classical result in the literature on differential privacy that the histogram mechanism in Fig. 1b is 1-DP but not ϵ-DP for $\epsilon < 1$ [31]. Intuitively, each query answer is perturbed independently by Laplace noise with scale 1 to mask the query answer, so the mechanism is 1-DP but not ϵ-DP for any $\epsilon < 1$.

Our approach: DP refutation via expectation mismatch. By the above definition, to refute ϵ-DP of P, it suffices to find a pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ and an output event A such that $\mathbb{P}_{\mathbf{x}^1_{in}}[Output(A)] > e^\epsilon \cdot \mathbb{P}_{\mathbf{x}^2_{in}}[Output(A)]$. Our method for DP refutation explicitly computes such a pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$. However, rather than also searching for an output event A, it

instead computes a non-negative function $f : \mathbb{R}^{|\mathcal{V}_{out}|} \to \mathbb{R}$ over PP outputs for which

$$\mathbb{E}_{\mathbf{x}^1_{in}}[f] > e^{\epsilon} \cdot \mathbb{E}_{\mathbf{x}^2_{in}}[f]. \tag{3}$$

Here, we use $\mathbb{E}_{\mathbf{x}^1_{in}}$ and $\mathbb{E}_{\mathbf{x}^2_{in}}$ to denote the expectation operators over the output distributions induced by the PP on these two inputs. We show in the proof of Theorem 4.6 that P is not ϵ-DP if and only if there exist a pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ and a non-negative function f over outputs for which eq. (3) holds. Hence, in order to refute ϵ-DP of the PP P, we may without loss of generality search for a pair of similar inputs $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ and such a non-negative function f over outputs.

Example 4. *Consider again the randomized response mechanism in Fig. 1a. Let $f(out) = out^2$ be a non-negative function over its output variable. Based on the probabilities in Table ??, one can observe that for two similar inputs $(x^1_{in}, out^1_{in}) = (1, 0)$ and $(x^2_{in}, out^2_{in}) = (0, 0)$, we have*

$$\mathbb{E}_{(1,0)}[f] = 0.75, \qquad \mathbb{E}_{(0,0)}[f] = 0.25.$$

Hence, we have $\mathbb{E}_{(1,0)}[f] = e^{\ln 3} \cdot \mathbb{E}_{(0,0)}[f]$, and for any $\epsilon < \ln 3$ it holds that $\mathbb{E}_{(1,0)}[f] > e^{\epsilon} \cdot \mathbb{E}_{(0,0)}[f]$. This refutes ϵ-DP of the mechanism for every $\epsilon < \ln 3$.

Next, consider the histogram mechanism in Fig. 1b. Already in the case when $n = 1$ (the simplified PP for $n = 1$ is showed at the bottom of Fig. 1b), if we consider two similar inputs $(q^1_{0\,in}, \eta^1_{in}) = (1.5, 0)$ and $(q^2_{0\,in}, \eta^2_{in}) = (2.5, 0)$ and a non-negative function $f(q_0) = 2q_0^2 + 393.92q_0^4$ over the output variable q_0, we observe that $\mathbb{E}_{(1.5,0)}[f] = 22092.64 \leq e^{\ln 2.46} \cdot 54402.08 = e^{\ln 2.46} \cdot \mathbb{E}_{(2.5,0)}[f]$, which implies that the mechanism is not ϵ-DP for any $\epsilon < \ln 2.46 \approx 0.9001$.

Sound and complete proof rule for DP refutation: Expectation super/sub-martingale witnesses. A challenge in computing such a function f is the exact computation of the expected value of f on PP output, since this expected value may not admit a closed-form expression in the general case (e.g. for PPs with unbounded loops [42]). To overcome this challenge, our method instead computes a non-negative function f over PP outputs *simultaneously and together with* a lower bound $L \leq \mathbb{E}_{\mathbf{x}^1_{in}}[f]$ and an upper bound $U \geq \mathbb{E}_{\mathbf{x}^2_{in}}[f]$ on the two expected values, while in addition requiring that $L > e^{\epsilon} \cdot U$. These three inequalities together entail the inequality in eq. (3) and hence allow our method to refute ϵ-DP.

Unlike the exact expected value of the function f on PP output which is in general hard to compute, upper and lower bounds on the expected value may be efficiently and automatically computed by using *upper expectation supermartingales (UESMs)* for upper bounds and *lower expectation submartingales (LESMs)* for lower bounds [19, 20]. A UESM (resp. LESM) for a function f over PP outputs is a function that assign a real value to each PP state, which is required to satisfy a set of conditions that together ensure that the UESM (resp. LESM) evaluates to an upper (resp. lower) bound on the expected value of f on PP output. It was shown in [20] that U/LESMs provide a sound and complete proof rule for deriving upper and lower bounds on the expected value of a function on PP output. Their name is due to their connection to super- and submartignale processes from probability theory, which are used to formally establish soundness and completeness results [61]. Our method leverages U/LESMs and we propose a new proof rule for refuting ϵ-DP in a PP which we call expectation super/sub-martingale witnesses. Informally, given a PP, a similarity relation Φ over its inputs and a privacy budget $\epsilon \geq 0$, an *expectation super/sub-martingale witness for ϵ-DP refutation* in the PP is a tuple $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, L_f, U_f)$ where:

(R1) $\mathbf{x}^1_{in}, \mathbf{x}^2_{in}$ is a pair of similar inputs, i.e. $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$;
(R2) $f : \mathbb{R}^{\mathcal{V}_{out}} \to \mathbb{R}$ is a non-negative function over PP outputs;
(R3) L_f is an LESM for f that satisfies the necessary LESM conditions and yields a lower expected value bound $L \leq \mathbb{E}_{\mathbf{x}^1_{in}}[f]$;

(R4) U_f is a UESM for f that satisfies the necessary UESM conditions and yields an upper expected value bound $U \geq \mathbb{E}_{\mathbf{x}^2_{in}}[f]$
(R5) The inequality $L > e^{\epsilon} \cdot U$ holds.

We formalize the notions of U/LESMs, the conditions that they need to satisfy, and our novel proof rule for ϵ-DP refutation in Section 4, and also illustrate it on our two running examples in Fig. 1a and Fig. 1b. Moreover, in Theorem 4.6, we prove that expectation super/sub-martingale witness for ϵ-DP refutation provide a *sound and complete* proof rule for refuting ϵ-DP in PPs. This is the main theoretical result of our work.

Automation: Template-based synthesis. We also present an algorithm for automated ϵ-DP refutation based on our novel sound and complete proof rule. Our algorithm considers arithmetic PPs in which all arithmetic expressions are assumed to be polynomials (see the full list of algorithm assumptions in Section 5) and it computes an instance of an expectation super/sub-martingale witness for ϵ-DP refutation that can be expressed in polynomial real arithmetic.

The key challenge in achieving this is the effective computation of all elements of the witness tuple $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, L_f, U_f)$. Note that the elements of the tuple cannot be computed separately, and the computation must be guided by the requirement that the LESM L_f and the UESM U_f give rise to lower and upper bounds L and U on the expected value of function f on output that satisfy $L > e^{\epsilon} \cdot U$. So, we propose an algorithm that computes these five objects simultaneously while also imposing this inequality on the computation process, by following a template-based synthesis approach. Our algorithm fixes a symbolic template for each of the elements of the witness tuple $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, L_f, U_f)$, where templates for f, L_f and U_f are defined in terms of polynomial expressions over real-valued symbolic template variables. It then collects a system of constraints over the symbolic template variables which together entail that $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, L_f, U_f)$ is a valid expectation super/sub-martingale witness for ϵ-DP refutation. The collected system of constraints is processed by applying results from real algebraic geometry, in order to simplify its solving. Finally, the algorithm uses an off-the-shelf SMT solver to solve the system of constraints, and any solution gives rise to a valid proof of ϵ-DP refutation. We present the details of our template-based synthesis algorithm in Section 5. Moreover, in Theorem 5.1, we show that the algorithm is sound, conditionally semi-complete (a notion that we formalize in Section 5), and has a PSPACE upper bound on its complexity.

Novelty and limitations. As discussed in Section 1, our theoretical results give rise to a sound and complete proof rule for ϵ-DP refutation that allows fully automated reasoning. Our experiments in Section 6 also demonstrate the ability of our automated method to refute ϵ-DP for a large number of challenging stochastic mechanisms and superior practical performance compared to the state of the art. This is the main novelty of our work. However, our method still suffers from a few limitations which we discuss below, addressing which is an exciting venue for future work:

(1) *Theory: Restriction to ϵ-DP refutation.* Our method considers an established notion of ϵ-DP and the problem of its refutation. However, another popular notion of DP is (ϵ, δ)-DP [33]. Given a PP, a similarity relation Φ over its inputs and $\epsilon, \delta \geq 0$, a PP is said to be (ϵ, δ)-DP if for every pair of similar inputs $\mathbf{x}^1_{in} \sim_{\Phi} \mathbf{x}^2_{in}$ and for every output event A we have $\mathbb{P}_{\mathbf{x}^1_{in}}[Output(A)] \leq e^{\epsilon} \cdot \mathbb{P}_{\mathbf{x}^2_{in}}[Output(A)] + \delta$. Hence, (ϵ, δ)-DP is a more general notion of differential privacy with ϵ-DP being the special case when $\delta = 0$.
Our method for ϵ-DP refutation *does not* readily extend to (ϵ, δ)-DP refutation. This is because the immediate extension of DP refutation via expectation mismatch reasoning as in eq. (3) becomes unsound in the case of (ϵ, δ)-DP refutation. Studying how to extend this style of reasoning to (ϵ, δ)-DP refutation and other notions of privacy is an interesting direction of future work. We also note that this restriction is not limited to our method, and most existing

methods for formal and automated DP refutation are also restricted to ϵ-DP refutation while not being applicable to (ϵ, δ)-DP refutation [9, 10, 39, 59].

(2) *Automation: Restriction to polynomial arithmetic PPs.* Our novel proof rule and its soundness and completeness guarantees apply to general PPs in which arithmetic expressions may be arbitrary Borel-measurable functions. However, our template-based synthesis algorithm for automated computation of expectation super/sub-martingale witnesses is restricted to polynomial arithmetic PPs. Considering algorithms for DP refutation in non-polynomial PPs would be an interesting direction of future work. Note, however, that our class of polynomial arithmetic PPs still allows sampling instructions from non-polynomial distributions such as Laplacian or normal, and we only require that arithmetic expressions appearing in program variable assignments are polynomial expressions.

3 Preliminaries

We use boldface notation to denote vectors, e.g. $\mathbf{x}$ or $\mathbf{y}$. For a real-valued vector $\mathbf{x} \in \mathbb{R}^n$ and an index $1 \leq i \leq n$, we use $\mathbf{x}[i]$ to denote the value of the i-th component of $\mathbf{x}$. In what follows, we assume familiarity with basic notions of probability theory, such as *probability space*, *random variable*, or *expected value*. We use $\mathcal{D}(A)$ to define the set of all probability distributions over the set A. We refer the reader to [61] for formal definitions of these notions.

3.1 Probabilistic Programs

We formalize the problem of DP refutation in the setting of *probabilistic programs (PPs)*. We consider imperative arithmetic PPs that allow standard programming constructs, such as program variable assignments, if-branching, sequential composition and loops. In addition, we allow constructs for *sampling instructions* which sample a value from a probability distribution and assign it to some program variable. These are denoted by **sample**(...) commands in our syntax. We allow sampling from both discrete and continuous probability distributions. Note that probabilistic branching instructions **if prob**(...) can be obtained as syntactic sugar, by using a sampling instruction from a Bernoulli distribution followed by conditional branching. Example PPs that will serve as our running examples for DP analysis are shown in Fig. 1a and Fig. 1b.

We assume that all program variables in our PPs are real-valued. Moreover, for the semantics of our PPs to be mathematically well defined, we assume that every arithmetic expression appearing in our PPs defines a *Borel-measurable* function over program variables. Borel-measurability is a standard assumption in the PP analysis literature, and allows most standard arithmetic operations and functions from mathematical analysis [61].

Probabilistic control flow graphs. We use *probabilistic control-flow graphs (pCFGs)* [3, 15, 19] to formally model our PPs. A pCFG is a tuple $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$, where:

- L is a finite set of *locations*;
- $\mathcal{V} = \{x_1, \ldots, x_{|\mathcal{V}|}\}$ is a finite set of *program variables*;
- $\mathcal{V}_{out} = \{x_1, \ldots, x_{|\mathcal{V}_{out}|}\} \subseteq \mathcal{V}$ is a finite set of *output variables*;
- $\ell_{in} \in L$ is the *initial program location*;
- $\theta_{in} \subseteq \mathbb{R}^{|\mathcal{V}|}$ is the set of *initial variable valuations*;
- $\mapsto \subseteq L \times \mathcal{D}(L)$ is a finite set of *transitions*. For each transition $\tau = (\ell, Pr)$, ℓ is its *source location* and $Pr : L \to [0, 1]$ is a probability distribution over *successor locations*;
- G is a map assigning to each transition $\tau = (\ell, Pr) \in \mapsto$ a predicate $G(\tau)$ over $\mathcal{V}$ specifying if τ can be executed, which we call a *guard*;
- Up is a map assigning to each transition $\tau = (\ell, Pr) \in \mapsto$ an *update* $Up(\tau) = (j, u)$, where $j \in \{1, \ldots, |\mathcal{V}|\}$ is a *variable index* and u is an *update element* which can be either

- the bottom element $u = \bot$, denoting no variable update, or
- a Borel-measurable arithmetic expression $u : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}$, or
- a probability distribution $u = \delta$, denoting that variable value is sampled according to δ.

The translation of PPs into pCFGs is standard, hence we omit the details and refer the reader to e.g. [3]. We make the following assumptions about our pCFGs:

(i) A pCFG C contains a special *terminal location* ℓ_{out} which only has one outgoing self-loop transition τ with $G(\tau) \equiv$ true and the bottom update element.
(ii) Each location ℓ has at least one outgoing transition and $\bigvee_{\tau=(l,\_)} G(\tau) \equiv \mathit{true}$.
(iii) The guards of any two distinct transitions τ_1 and τ_2 outgoing from the same location ℓ are *mutually exclusive*, i.e. $G(\tau_1) \wedge G(\tau_2) \equiv \mathit{false}$

The first two assumptions are to avoid deadlock situations where the program cannot progress, and are imposed without loss of generality as they can be enforced by introducing additional transitions. The last assumption prevents pCFGs from admitting non-deterministic behavior.

States, paths and runs. A *state* in C is a tuple $(\ell, \mathbf{x})$, where ℓ is a location and $\mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$ is a variable valuation. A transition $\tau = (\ell, Pr)$ is *enabled* at a state $(\ell, \mathbf{x})$, if $\mathbf{x} \models G(\tau)$. A state $(\ell', \mathbf{x}')$ is a *successor* of a state $(\ell, \mathbf{x})$, if there is an enabled transition $\tau = (\ell, Pr)$ at $(\ell, \mathbf{x})$ such that $Pr(\ell') > 0$ and $\mathbf{x}'$ can be obtained by applying the update of τ to $\mathbf{x}$. A state $(\ell, \mathbf{x})$ is said to be an *initial state*, if $\ell = \ell_{in}$ and $\mathbf{x} \in \theta_{in}$. A state $(\ell, \mathbf{x})$ is said to be a *terminal state*, if $\ell = \ell_{out}$.

A *finite path* in C is a sequence $(\ell_0, \mathbf{x}_0), (\ell_1, \mathbf{x}_1), \ldots, (\ell_k, \mathbf{x}_k)$ of states with $(\ell_0, \mathbf{x}_0)$ an initial state and each state $(\ell_{i+1}, \mathbf{x}_{i+1})$ being a successor of $(\ell_i, \mathbf{x}_i)$. A *run* (or an *execution*) in C is an infinite sequence of states whose each finite prefix is a finite path. We use Run^C to denote the set of all runs in C. A state $(\ell, \mathbf{x})$ is *reachable* in C if there exists a finite path in C whose last state is $(\ell, \mathbf{x})$.

Semantics. The pCFG semantics are formalized as (possibly infinite-state) discrete-time Markov chains [3]. In particular, a pCFG C together with an initial variable valuation $\mathbf{x}_{in} \in \theta_{in}$ define a discrete-time Markov chain over the set of states in C, whose trajectories correspond to runs in C. Each trajectory of the Markov chain starts in the initial state $(\ell_{in}, \mathbf{x}_{in})$. Then, at each time step, if the trajectory is in state $(\ell_i, \mathbf{x}_i)$, the next state $(\ell_{i+1}, \mathbf{x}_{i+1})$ is defined according to the probability distribution over successor states and the update of the unique pCFG transition τ_i enabled at $(\ell_i, \mathbf{x}_i)$. This Markov chain gives rise to a probability space $(Run^C, \mathcal{F}^C, \mathbb{P}^C_{\mathbf{x}_{in}})$ over the set of all runs in C that start in the initial state $(\ell_{in}, \mathbf{x}_{in})$. The probability space is formally defined via the cylinder construction [48]. We use $\mathbb{E}^C_{\mathbf{x}_{in}}$ to denote the expectation operator in this probability space.

Termination. We say that a run $\rho = (\ell_0, \mathbf{x}_0), (\ell_1, \mathbf{x}_1), \ldots$ in the pCFG C is *terminating*, if it reaches some terminal state. We use $Term \subseteq Run^C$ to denote the set of all terminating runs in C and use *TimeTerm* as the random variable for the number of steps C takes before termination. A pCFG C is said to be *almost-surely (a.s.) terminating*, if the probability of a random run terminating is equal to 1, i.e. if $\mathbb{P}_{\mathbf{x}_{in}}[Term] = 1$ for every initial variable valuation $\mathbf{x}_{in} \in \theta_{in}$. Our algorithm for refuting DP assumes that the underlying pCFG terminates almost-surely.

3.2 Differential Privacy

We now formally define the notion of differential privacy. Consider a PP with some specified set of inputs and a specified set of output variables. Intuitively, a PP is differentially private if, for any two "similar" input variable valuations $\mathbf{x}^1_{in}$ and $\mathbf{x}^2_{in}$, the probability of every output event on these two inputs differs by an insignificant amount. For example, suppose that we define the notion of similarity of inputs by the presence and absence of exactly one datapoint in the input. Then, a PP being differentially private ensures that an adversarial observer cannot easily detect information about individual datapoints by observing outputs on similar inputs.

In order to formalize the above intuition, we need to formalize the notions of input similarity and of output distributions. Consider an a.s. terminating pCFG $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$. We say that the set of initial variable valuations $\theta_{in} \subseteq \mathbb{R}^{|\mathcal{V}|}$ is the set of its *inputs*, and that the set of all output variable valuations $\mathbb{R}^{\mathcal{V}_{out}}$ is the set of its *outputs*.

Similarity of inputs. To formally reason about the similarity of inputs, we consider a boolean predicate $\Phi(\mathbf{x}^1_{in}, \mathbf{x}^2_{in})$ over $\mathbb{R}^{\mathcal{V}} \times \mathbb{R}^{\mathcal{V}}$ which we call a *similarity relation*. We say that two inputs $\mathbf{x}^1_{in}, \mathbf{x}^2_{in} \in \theta_{in}$ are *similar* if $\Phi(\mathbf{x}^1_{in}, \mathbf{x}^2_{in})$ is true, and write $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$.

Output distributions. For each input $\mathbf{x}_{in} \in \theta_{in}$, an a.s. terminating pCFG defines a probability distribution over the set of its outputs as follows. Denote by $\mathbf{x}^{out}$ the projection of a variable valuation $\mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$ to the components of output variables in $\mathcal{V}_{out}$. For a Borel-measurable set $A \subseteq \mathbb{R}^{|\mathcal{V}_{out}|}$, define the set of terminating pCFG runs that reach a variable valuation in A upon termination via $Output(A) = \{\rho \in Run^C \mid \rho \text{ reaches a terminal state } (\ell_{out}, \mathbf{x}) \text{ with } \mathbf{x}^{out} \in A\}$.

The *output distribution* of C given an initial variable valuation $\mathbf{x}_{in} \in \theta_{in}$ is a probability distribution over outputs $\mathbb{R}^{|\mathcal{V}_{out}|}$ that is defined via $\mu^C_{\mathbf{x}_{in}}(A) = \mathbb{P}^C_{\mathbf{x}_{in}}[Output(A)]$, for every Borel-measurable set $A \subseteq \mathbb{R}^{|\mathcal{V}_{out}|}$. We omit the superscript C whenever it is clear from the context.

Definition 3.1 (Differential privacy [31, 33]). Consider a pCFG $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ which is assumed to be a.s. terminating, a similarity relation Φ over its inputs and $\epsilon \geq 0$. We say that the pCFG C is *ϵ-differentially private (ϵ-DP)*, if for every pair of inputs $\mathbf{x}^1_{in}, \mathbf{x}^2_{in} \in \theta_{in}$ such that $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ and for every Borel-measurable set of outputs $A \subseteq \mathbb{R}^{|\mathcal{V}_{out}|}$ we have

$$\mathbb{P}_{\mathbf{x}^1_{in}}\Big[Output(A)\Big] \leq e^\epsilon \cdot \mathbb{P}_{\mathbf{x}^2_{in}}\Big[Output(A)\Big], \tag{4}$$

or, equivalently, $\mu_{\mathbf{x}^1_{in}}(A) \leq e^\epsilon \cdot \mu_{\mathbf{x}^2_{in}}(A)$.

Problem statement. We now define the DP refutation problem that we consider in this work. Suppose that $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ is a pCFG, Φ is a similarity relation over its inputs and $\epsilon \geq 0$. Prove that C is not ϵ-DP, and compute a pair of inputs $\mathbf{x}^1_{in}$ and $\mathbf{x}^2_{in}$ such that $\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}$ but for which eq. (4) is violated.

Remark 1. *Note that we use the set-based definition of DP which relates probabilities of events instead of single elements. This is because we are considering PPs with real-valued variables and continuous distributions, which may make the probability of observing a single output element zero.*

4 Sound and Complete Proof Rule for Differential Privacy Refutation

We now present our novel proof rule for DP refutation. As outlined in Section 2, our proof rule is based on the notions of upper expectation supermartingales and lower expectation submartingales, which respectively provide proof rules for computing upper and lower bounds on the expected value of a function upon PP termination. To that end, in Section 4.1 we first recall these notions. We then present our proof rule for DP refutation and establish its soundness and completeness in Section 4.2. Let $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ be a pCFG. We fix some additional terminology:

(1) A *state function* η is a function which to each pCFG location $\ell \in L$ assigns a Borel-measurable function $\eta(\ell) : \mathbb{R}^{\mathcal{V}} \to \mathbb{R}$ over program variables. We use $\eta(\ell, \mathbf{x})$ for $\eta(\ell)(\mathbf{x})$ as well.
(2) A *predicate function* Π is a function which to each pCFG location $\ell \in L$ assigns a predicate $\Pi(\ell)$ over program variables. The predicate function Π naturally induces a set of states $\{(\ell, \mathbf{x}) \mid \mathbf{x} \models \Pi(\ell)\}$ and we also use Π to denote this set.

4.1 Background: Expectation Supermartingales and Submartingales

Recall from Section 2 that, given a function f over pCFG outputs, upper expectation supermartingales (UESMs) and lower expectation submartingales (LESMs) for f are functions which assign a real value to each pCFG state and which are required to satisfy a set of conditions at reachable pCFG states. These conditions together ensure that U/LESMs give rise to an upper/lower bound on the expected value of f on pCFG output.

The following definitions formalize the notions of U/LESMs. However, since it is in general not feasible to compute the set of all reachable pCFG states, we impose the defining conditions of U/LESMs over a supporting invariant rather than over reachable pCFG states. Given a pCFG C, an *invariant* in C is a predicate function I which contains all reachable states in the pCFG, i.e. for every reachable state $(\ell, \mathbf{x})$ we have $\mathbf{x} \models I(\ell)$. This is done with later automation in mind, since invariants can be synthesized by using off-the-shelf tools as we discuss in Section 5.

Definition 4.1 (Upper expectation supermartingale [19]). Let $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ be an a.s. terminating pCFG, I be an invariant in C, and $f\colon \mathbb{R}^{\mathcal{V}_{out}} \to \mathbb{R}$ be a Borel-measurable function over the outputs in C. An *upper expectation supermartingale (UESM)* for f is a state function U_f such that the following conditions are satisfied:

- *Zero on output.* For every $\mathbf{x} \models I(\ell_{out})$, it holds that $U_f(\ell_{out}, \mathbf{x}) = 0$.
- *Expected f-decrease.* For every location ℓ, transition $\tau = (\ell, Pr) \in \mapsto$ and variable valuation $\mathbf{x} \models I(\ell) \wedge G(\tau)$, if $\mathbf{x}'$ is the valuation obtained by performing τ, it holds that

$$U_f(\ell, \mathbf{x}) + f(\mathbf{x}^{out}) \geq \sum_{\ell' \in L} Pr(\ell') \cdot \mathbb{E}[U_f(\ell', \mathbf{x}') + f(\mathbf{x}'^{out})].$$

Definition 4.2 (Lower expectation submartingale [19]). Let $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ be an a.s. terminating pCFG, I be an invariant in C, and $f\colon \mathbb{R}^{\mathcal{V}_{out}} \to \mathbb{R}$ be a Borel-measurable function over the outputs in C. A *lower expectation submartingale (LESM)* for f is a state function L_f such that the following conditions are satisfied:

- *Zero on output.* For every $\mathbf{x} \models I(\ell_{out})$, it holds that $L_f(\ell_{out}, \mathbf{x}) = 0$.
- *Expected f-increase.* For every location ℓ, transition $\tau = (\ell, Pr) \in \mapsto$ and variable valuation $\mathbf{x} \models I(\ell) \wedge G(\tau)$, if $\mathbf{x}'$ is the valuation obtained by performing τ, it holds that

$$L_f(\ell, \mathbf{x}) + f(\mathbf{x}^{out}) \leq \sum_{\ell' \in L} Pr(\ell') \cdot \mathbb{E}[L_f(\ell', \mathbf{x}') + f(\mathbf{x}'^{out})].$$

Intuitively, if U_f (resp. L_f) is a UESM (resp. LESM) for a function f over the outputs in C, the value of $U_f(\ell, \mathbf{x}) + f(\mathbf{x}^{out})$ (resp. $L_f(\ell, \mathbf{x}) + f(\mathbf{x}^{out})$) is zero on output and for every $\mathbf{x} \models I(\ell)$ it decreases (resp. increases) in expectation upon every one-step execution of the pCFG.

Unfortunately, the Zero on output and the Expected f-decrease/increase conditions alone are not sufficient for U/LESMs to provide sound proof rules for establishing upper or lower bounds on the expected value of a function on program output. However, it was shown in [19] that soundness can be ensured by additionally imposing one of the four *OST-soundness* conditions on the program and the U/LESM pair. The name of these conditions is derived from the Optional Stopping Theorem (OST) in martingale theory, which lies at the core of the soundness proof. The first three conditions are derived from the classical OST [61], whereas the fourth condition is derived from the extended OST [57]. In what follows, we first provide an intuition behind the four OST-soundness conditions, after which we formally define them in Definition 4.3:

(C1) **Bounded Termination Time:** The program terminates within a fixed constant number of steps, effectively capping the execution length.

(C2) **Bounded Values:** The sum of the state functions that define U/LESM and the target function f is bounded by a fixed constant throughout all possible program runs.

(C3) **Bounded Expected Changes:** The program terminates in finite expected time, and the expected step-wise change of the sum of the state functions that define U/LESM and the target function f is bounded by a fixed constant throughout all possible program runs.

(C4) **Probabilistic Termination with Polynomial Step Bounds:** The probability of the program running for a long time decreases exponentially, and the worst-case step-wise change of the sum of the state functions that define U/LESM and the target function f grows at most polynomially in the number of execution steps.

Definition 4.3 (OST-soundness [19]). Let C be a pCFG, η be a state function in C and $f : \mathbb{R}^{\mathcal{V}}_{out} \to \mathbb{R}$ be Borel-measurable. Moreover, let ℓ_i be the i-th location and $\mathbf{x}_i$ variable valuation along a random run ρ in C and $Y_i(\rho) = \eta(\ell_i, \mathbf{x}_i) + f(\mathbf{x}_i^{out})$. The triple (C, η, f) is said to be *OST-sound* if $\mathbb{E}[|Y_i|] < \infty$ holds for every $i \geq 0$, and at least one of the following conditions holds for every $\mathbf{x}_{in} \in \theta_{in}$:

(C1) $\mathbb{P}_{\mathbf{x}_{in}}[\mathit{TimeTerm} < c] = 1$ for a constant c, i.e. C has uniformly bounded termination time.

(C2) There is a constant c such that, for each time step $i \geq 0$ and every run ρ in C, it holds that $|Y_i(\rho)| < c$, i.e. the sum of η and f is uniformly bounded over program runtime.

(C3) $\mathbb{E}_{\mathbf{x}_{in}}[\mathit{TimeTerm}] < \infty$ and there exists a constant c such that for each time step $i \geq 0$ and every run ρ in C, it holds that $\mathbb{E}[|Y_{i+1} - Y_i| \mid \mathcal{F}_i](\rho) < c$, i.e. the expected one-step change of Y_i when conditioned on the past steps is uniformly bounded over program runtime.

(C4) There exist real values M, c_1, c_2, d, such that (i) for all $n \in \mathbb{N}$ that are sufficiently large, $\mathbb{P}_{\mathbf{x}_{in}}[\mathit{TimeTerm} > n] \leq c_1 \cdot e^{-c_2 \cdot n}$, and (ii) for each time step $i \geq 0$ and every run ρ in C it holds that $|Y_{i+1}(\rho) - Y_i(\rho)| \leq M \cdot n^d$.

The following theorem shows that U/LESMs for a function f provide sound proof rules for computing upper/lower bounds on the expected value of the function f on pCFG output.

Theorem 4.4 (Soundness of U/LESMs [19]). *Let $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ be an a.s. terminating pCFG, I be an invariant in C, $f : \mathbb{R}^{\mathcal{V}_{out}} \to \mathbb{R}$ be a Borel-measurable function over the outputs in C, U_f be an UESM for f and L_f be an LESM for f. If the triples (C, U_f, f) and (C, L_f, f) are OST-sound, then*

$$U_f(\ell_{in}, \mathbf{x}_{in}) + f(\mathbf{x}_{in}^{out}) \geq \mathop{\mathbb{E}}_{\mathbf{x} \sim \mu_{\mathbf{x}_{in}}}[f(\mathbf{x}^{out})]$$

$$L_f(\ell_{in}, \mathbf{x}_{in}) + f(\mathbf{x}_{in}^{out}) \leq \mathop{\mathbb{E}}_{\mathbf{x} \sim \mu_{\mathbf{x}_{in}}}[f(\mathbf{x}^{out})]$$

The works [19, 20] show that U/LESMs are not only sound, but also complete proof rule for deriving tight upper and lower bounds on the expected value of a function f on output. Moreover, the computation of U/LEMSs and checking of OST-soundness conditions can be fully automated, which we will use later in Section 5 when designing our automated algorithm for ϵ-DP refutation.

Example 5 (U/LESM for RR-1). *Consider the randomized response mechanism in Fig. 1a. The following table shows an example of a UESM U_f and an LESM L_f for $f(out) = out^2$:*

Label	*UESM*	*LESM*
l_1	$0.25 - out^2 + 0.5x^2$	$0.25 - out^2 + 0.5x^2$
l_2	$-out^2 + x^2$	$-out^2 + x^2$
l_3	$0.5 - out^2$	$0.5 - out^2$
l_4	$-out^2$	$-out^2$
l_5	$1 - out^2$	$1 - out^2$
l_t	0	0

The mechanism trivially satisfies the OST-soundness condition (C1) in Definition 4.3, hence the U/LESM define OST-sound triples. Moreover, the above U/LESM give rise to tight upper and lower bounds on the expected value of f upon termination.

4.2 Proof Rule for Differential Privacy Refutation

We are now ready to present our proof rule for ϵ-DP refutation. Consider a pCFG C, a similarity relation Φ over its inputs, and a privacy budget $\epsilon \geq 0$. To refute ϵ-DP of the pCFG, i.e. to prove that C is *not* ϵ-DP, we need to find a pair of similar inputs $\mathbf{x}_{in}^1 \sim_\Phi \mathbf{x}_{in}^2$ and an event over outputs $A \subseteq \mathbb{R}^{\mathcal{V}_{out}}$ such that $\mathbb{P}_{\mathbf{x}_{in}^1}[Output(A)] > e^\epsilon \cdot \mathbb{P}_{\mathbf{x}_{in}^2}[Output(A)]$. As outlined in Section 2, our refutation proof rule explicitly characterizes two similar inputs $\mathbf{x}_{in}^1 \sim_\Phi \mathbf{x}_{in}^2$. However, instead of also explicitly characterizing a set of outputs A, it characterizes a witness of the existence of such a set A in the form of a non-negative function over the pCFG outputs and a U/LESM pair.

Definition 4.5 (Expectation super/sub-martingale witness for ϵ-DP refutation). Given a pCFG $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$, a similarity relation $\sim_\Phi$ on θ_{in}, a privacy budget $\epsilon \geq 0$, and an invariant I in C, an *expectation super/sub-martingale witness for ϵ-DP refutation* in C is a tuple $(\mathbf{x}_{in}^1, \mathbf{x}_{in}^2, f, U_f, L_f)$, where:

(R1) $\mathbf{x}_{in}^1, \mathbf{x}_{in}^2 \in \theta_{in}$ is a pair of similar inputs, i.e. $\mathbf{x}_{in}^1 \sim_\Phi \mathbf{x}_{in}^2$;
(R2) $f : \mathbb{R}^{\mathcal{V}_{out}} \to \mathbb{R}$ is a non-negative Borel-measurable function over pCFG outputs;
(R3) L_f is an LESM for f in C such that the triple (C, L_f, f) is OST-sound;
(R4) U_f is a UESM for f in C such that the triple (C, U_f, f) is OST-sound;
(R5) $L_f(\ell_{in}, \mathbf{x}_{in}^1) + f(\mathbf{x}_{in}^{1\,out}) > e^\epsilon \left(U_f(\ell_{in}, \mathbf{x}_{in}^2) + f(\mathbf{x}_{in}^{2\,out})\right)$.

Intuitively, our proof rule consists of two similar input points $\mathbf{x}_{in}^1 \sim_\Phi \mathbf{x}_{in}^2$, a non-negative function f over outputs, and a witness to show that the expectation of f upon the termination of C on input $\mathbf{x}_{in}^1$ is more than a factor of e^ϵ larger than its expectation upon the termination of C on input $\mathbf{x}_{in}^2$. This discrepancy between the output distributions on two inputs is witnessed by an LESM L_f and a UESM U_f for f in C which satisfy the inequality $L_f(\ell_{in}, \mathbf{x}_{in}^1) + f(\mathbf{x}_{in}^{1\,out}) > e^\epsilon \left(U_f(\ell_{in}, \mathbf{x}_{in}^2) + f(\mathbf{x}_{in}^{2\,out})\right)$. The following theorem establishes that the expectation super/sub-martingale witnesses indeed provide a sound and complete proof rule for refuting ϵ-DP.

Theorem 4.6 (Soundness and completeness, Proof in the extended version [21]). *Given a pCFG $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$, a similarity relation $\sim_\Phi$ on θ_{in}, a privacy budget $\epsilon \geq 0$, and an invariant I in C, the pCFG C is* not ϵ-DP if and only if *there exists an expectation super/sub-martingale witness for ϵ-DP refutation $(\mathbf{x}_{in}^1, \mathbf{x}_{in}^2, f, U_f, L_f)$.*

Proof Sketch. For soundness, we show that if C is ϵ-DP, then for every non-negative function f and similar inputs x_1, x_2, it holds that $\mathbb{E}_{\mathbf{x}\sim\mu_{x_1}}[f(\mathbf{x}^{out})] \leq e^\epsilon \mathbb{E}_{\mathbf{x}\sim\mu_{x_2}}[f(\mathbf{x}^{out})]$. Hence, if an expectation super/sub-martingale witness exists, then C is not ϵ-DP, due to Theorem 4.4.

For completeness, we show that if C is not ϵ-DP, i.e. a pair of similar inputs $x_1 \sim_\Phi x_2$ and set of outputs A exist such that $\mu^C_{\mathbf{x}_{in}^1}(A) > e^\epsilon \mu^C_{\mathbf{x}_{in}^2}(A)$, then the indicator function of A can be extended to an expectation super/sub-martingale witness for ϵ-DP refutation of C. □

Note that non-negativity of f in condition (R2) is necessary for the soundness of our proof rule as shown in the extended version of the paper [21].

Example 6 (Witness for ϵ-DP refutation of RR-1). *Consider the randomized response mechanism in Fig. 1a. The tuple $\big((1,0),(0,0), f, U_f, L_f\big)$, with f, U_f and L_f defined as in Example 5, is an expectation super/sub-martingale witness for ϵ-DP refutation for every $\epsilon < \ln 3$, since:*

(R1) $(1,0) \sim_\Phi (0,0)$ by the definition of the similarity relation in Fig. 1a.
(R2) $f(out) = out^2$ is non-negative and Borel-measurable.
(R3) L_f is an LESM for f in C and (C, L_f, f) is OST-sound as shown in Example 5.
(R4) U_f is a UESM for f in C and (C, U_f, f) is OST-sound as shown in Example 5.
(R5) $L_f\big(\ell_{in}, (1,0)\big) + f(0) = 0.75$ and $U\big(\ell_{in}, (0,0)\big) + f(0) = 0.25$, hence (R5) holds for every $\epsilon < \ln 3$.

5 Automated Algorithm for Differential Privacy Refutation

We now present our algorithm for automated ϵ-DP refutation, by synthesizing an instance of our expectation super/sub-martingale witness that we introduced in Section 4. Since the problem of verification (and hence refutation) of DP properties is known to be undecidable [6], we cannot hope for a sound and complete algorithm that is applicable to all PPs. Therefore, we aim for a sound and conditionally semi-complete algorithm. By soundness, we mean that our algorithm provides formal guarantees on the correctness of its output. Conditional semi-completeness means that the algorithm is guaranteed to refute DP for a *subclass of non-DP PPs* which we formally characterize.

Algorithm assumptions. Given the undecidable nature of the DP refutation problem, our algorithm imposes several assumptions towards enabling its automation:

(1) *Polynomial programs.* We consider PPs and corresponding pCFGs in which all arithmetic expressions are polynomial expressions over program variables. Furthermore, by introducing dummy variables for expressions appearing in transition guards, we assume that arithmetic expressions appearing in transition guards are linear. The latter assumption is without loss of generality and is imposed for more efficient quantifier elimination (see details below).
(2) *Finite and computable moments of sampling instructions.* We assume that all sampling instructions appearing in our PPs have finite and computable moments. That is, for each probability distribution δ appearing in a sampling instruction and for each $p \in \mathbb{N}$, the p-th moment $\mathbb{E}_{X\sim\delta}[|X|^p]$ is finite and can be computed by our algorithm. This assumption holds by default for most standard probability distributions such as Laplace, Exponential, Normal or Uniform distribution, for which look-up tables for moment values are readily available.
(3) *Polynomial similarity relation.* We assume that the similarity relation $\Phi(\mathbf{x}^1_{in}, \mathbf{x}^2_{in})$ can be expressed by a conjunction of polynomial inequalities over variable valuations $\mathbf{x}^1_{in}$ and $\mathbf{x}^2_{in}$.
(4) *Linear supporting invariants.* Recall, in Definitions 4.1 and 4.2, we defined U/LESMs with respect to supporting invariants. Our algorithm assumes that such invariants are provided as $I(\ell)$ for every $\ell \in L$, and furthermore that $I(\ell)$ is given as a conjunction of linear inequalities over program variables. In practice, and in our prototype implementation, linear supporting invariants can be efficiently and automatically computed by off-the-shelf invariant generation tools for linear and polynomial programs [14, 35]. When synthesizing invariants for PPs, we treat sampling instructions as non-deterministic assignments.

Algorithm outline: Template-based synthesis of expectation super/sub-martingale witnesses. Our algorithm takes as input a pCFG $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ with polynomial arithmetic expressions and sampling instructions that have finite and computable moments, a linear invariant I, a polynomial similarity relation Φ, and a privacy budget $\epsilon \geq 0$. In addition, it takes as input a maximal polynomial degree parameter $D \in \mathbb{N}$ to be used in the synthesis algorithm.

In order to refute ϵ-DP of the pCFG C, the algorithm computes an instance $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, U_f, L_f)$ of an expectation super/sub-martingale witness for ϵ-DP refutation. This is done by following the template-based synthesis approach to simultaneously compute each of the five elements of the tuple. The synthesis proceeds in four steps.

Step 1: Setting up templates. The first step of the algorithm consists of setting up symbolic templates for each element of the tuple $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, U_f, L_f)$. For two inputs $\mathbf{x}^1_{in}$ and $\mathbf{x}^2_{in}$, the symbolic templates are defined by introducing two vectors of symbolic variables $\mathbf{x}^1_{in} = (\mathbf{x}^1_{in}[1], \ldots, \mathbf{x}^1_{in}[|\mathcal{V}|])$ and $\mathbf{x}^2_{in} = (\mathbf{x}^2_{in}[1], \ldots, \mathbf{x}^2_{in}[|\mathcal{V}|])$ which denote the values of each program variable on the two inputs. The template for the function f over pCFG outputs is defined by introducing a symbolic polynomial function over the output variables of degree at most D, i.e. $f(\mathbf{x}^{out}) = \sum_{m\in M^D(\mathcal{V}_{out})} c_m \cdot m$, where $M^D(\mathcal{V})$ is the set of all monomials of degree at most D over $\mathcal{V}_{out}$ and c_m's are the real-valued symbolic template variables. Finally, the templates for the UESM U_f and the LESM L_f are defined

as symbolic polynomial functions over the program variables for each location $\ell \in L$ in the pCFG:

$$U_f(\ell, \mathbf{x}) = \begin{cases} \sum_{m \in M^D(\mathcal{V})} s_{\ell,m} \cdot m & \ell \neq \ell_{out} \\ 0 & \ell = \ell_{out} \end{cases}, \qquad L_f(\ell, \mathbf{x}) = \begin{cases} \sum_{m \in M^D(\mathcal{V})} t_{\ell,m} \cdot m & \ell \neq \ell_{out} \\ 0 & \ell = \ell_{out} \end{cases}, \tag{5}$$

where $M^D(\mathcal{V})$ is the set of all monomials of degree at most D defined over $\mathcal{V}$ and $s_{\ell,m}$'s and $t_{\ell,m}$'s are the real-valued symbolic template variables. We explicitly require both the UESM and the LESM to be equal to 0 for $\ell = \ell_{out}$ as this is required by the Zero on output condition of UESMs and LESMs, see Definition 4.1 and Definition 4.2.

The values of the symbolic template variables at this stage are unknown. The later steps of the algorithm will then compute concrete values for these symbolic variables, which will together give rise to an instance of our ϵ-DP refutation witness.

Step 2: Constraint collection. In this step, the algorithm collects a system of constraints over the symbolic template variables which together entail that the tuple $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, U_f, L_f)$ defines a valid expectation super/sub-martingale witness for ϵ-DP refutation. To ensure this, we need to collect constraints for the five defining conditions in Definition 4.5:

(R1) The two inputs $\mathbf{x}^1_{in}$ and $\mathbf{x}^2_{in}$ need to satisfy the similarity relation. This is captured by the constraint $[\mathbf{x}^1_{in} \sim_\Phi \mathbf{x}^2_{in}]$. The constraint is collected by substituting the symbolic template variables that define $\mathbf{x}^1_{in}$ and $\mathbf{x}^2_{in}$ into the conjunction of polynomial inequalities that define the similarity relation Φ (recall that this was one of the algorithm assumptions).

(R2) f must be non-negative on all reachable output evaluations of C. This is captured by the constraint $[\forall \mathbf{x} \in \mathbb{R}^{\mathcal{V}}. \mathbf{x} \in I(\ell_{out}) \Rightarrow f(\mathbf{x}^{out}) \geq 0]$.

(R3) L_f must be an LESM for f and the triple (C, L_f, f) must be OST-sound. For L_f to be an LESM for f, it should satisfy the Zero on output and the Expected f-increase conditions in Definition 4.2. The former is satisfied by the definition of L_f in eq. (5). For the latter, for each $\ell \in L$ and each transition $\tau = (\ell, \Pr)$, the algorithm collects the following constraint:

$$\forall \mathbf{x} \in \mathbb{R}^{\mathcal{V}}. \mathbf{x} \in I(\ell) \cap G(\tau) \Rightarrow L_f(\ell, \mathbf{x}) + f(\mathbf{x}^{out}) \leq \sum_{\ell' \in L} Pr(\ell') \mathbb{E}[L_f(\ell', \mathbf{x}') + f(\mathbf{x}'^{out})] \tag{6}$$

where every occurrence of f and L_f is substituted by their symbolic polynomial template. Note that the symbolic expression for the expectation can be explicitly computed by the linearity of the expectation operator and since all moments of the sampling instructions are assumed to be finite and computable.

The system of constraints that entail that the triple (C, L_f, f) is OST-sound is analogous to the one constructed in the work [19] on automated computation of UESMs and LESMs. Hence, for the interest of space, we omit the details here and refer the reader to the extended version of the paper [21].

(R4) U_f must be a UESM for f and the triple (C, U_f, f) must be OST-sound. Similarly as above, U_f is zero on output by definition in eq. (5). For the expected f-decrease condition, the algorithm collects the same constraint as in eq. (6), and replaces $\leq$ with $\geq$ and L_f with U_f.

(R5) Finally, the algorithm collects the following constraint where f, U_f and L_f are substituted by their symbolic polynomial templates: $L_f(\ell_{in}, \mathbf{x}^1_{in}) + f(\mathbf{x}^{1\,out}_{in}) > e^\epsilon \cdot \left(U_f(\ell_{in}, \mathbf{x}^2_{in}) + f(\mathbf{x}^{2\,out}_{in})\right)$.

Step 3: Constraint simplification via $\forall$ quantifier elimination. Looking at the constraints collected in Step 2, we observe that the constraints for conditions (R1) and (R5) are purely existentially quantified polynomial constraints over the symbolic template variables. However, constraints for conditions (R2), (R3) and (R4) are of the form $[\forall \mathbf{x} \in \mathbb{R}^n. P(\mathbf{x}) \Rightarrow Q(\mathbf{x})]$, where P is a conjunction of linear inequalities over $\mathcal{V}$ (since we assume that all transition guards in the pCFG and the supporting invariant I are linear) and Q is a polynomial inequality over $\mathcal{V}$ (since we assume that all arithmetic

expressions in our pCFGs are polynomial). The algorithm then utilizes Handelman's Theorem [40] from algebraic geometry to translate each of these constraints into a purely existentially quantified set of linear inequalities over the symbolic template variables as well as auxiliary variables introduced by the translation. The translation is sound in the sense that every solution to the new system of constraints gives rise to a solution of the original system of constraints, while it is also complete under the additional condition that the supporting invariant I defines a closed and bounded set [4]. Since this is a standard procedure in template-based synthesis approaches to program analysis, we omit the details of this translation and refer the reader to [4, 19].

Step 4: Constraint solving. Denote by Ψ the resulting system of purely existentially quantified polynomial constraints obtained upon constraint collection in Step 2 and the application of universal quantifier elimination in Step 3. To synthesize an instance $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, U_f, L_f)$ of an expectation super/sub-martingale witness for ϵ-DP refutation, the algorithm solves the system of constraints Ψ by employing an off-the-shelf SMT solver. If the SMT solver finds a solution, the algorithm reports "ϵ-DP refuted" and returns the witness $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, U_f, L_f)$ to the user. Otherwise, the algorithm reports "Unknown".

Algorithm soundness, semi-completeness, complexity. The following theorem establishes soundness, conditional semi-completeness and an upper bound on the complexity of our algorithm. The soundness and conditional semi-completeness follow by soundness and semi-completeness of all steps of the algorithm. The complexity upper bound is due to the ϵ-DP refutation synthesis problem being reduced to an instance of existential theory of reals which can be done in PSPACE.

Theorem 5.1 (Soundness, conditional semi-completeness, complexity, Proof in the extended version [21]). *Let $C = (L, \mathcal{V}, \mathcal{V}_{out}, \ell_{in}, \theta_{in}, \mapsto, G, Up)$ be a pCFG, Φ a similarity relation over θ_{in}, $\epsilon \geq 0$ a privacy budget, and I an invariant in C that satisfy all the algorithm assumptions. The following claims hold:*

(1) (Soundness) *If the algorithm returns a tuple $T = (\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, U_f, L_f)$, then C is not ϵ-DP and T is a valid instance of an expectation super/sub-martingale witness for ϵ-DP refutation.*

(2) (Conditional Semi-completeness) *If C is not ϵ-DP, the invariant I defines a closed and bounded set, and there exists an instance of an expectation super/sub-martingale witness for ϵ-DP refutation $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, f, U_f, L_f)$ where f, L_f and U_f can be specified via polynomial expressions, then for a sufficiently large value of the maximum degree parameter D, the algorithm is guaranteed to find an instance of an expectation super/sub-martingale witness for ϵ-DP refutation.*

(3) (Complexity) *The runtime complexity of the algorithm is PSPACE in the size of the pCFG C, polynomial expressions that specify the similarity relation Φ, the invariant I and the privacy budget ϵ, for any fixed value of the maximum polynomial degree parameter D.*

Remark 2. *We emphasize two points regarding our algorithm:*

(1) Our algorithm is applicable to both discrete and continuous probabilistic programs. Specifically, in case the input variables of the program are integers, the system of constraints solved in Step 4 will become a mixed-integer system as $\mathbf{x}^1_{in}$ and $\mathbf{x}^2_{in}$ are required to be integers.

(2) Given a pCFG C and technical parameter D, our algorithm is fully automated. The invariant I can be generated automatically by state-of-the-art tools such as ASPIC[35] (used in our experiments). Additionally, our method could be paired with the template-based invariant generation method of [14] to synthesize the invariants and the UESM/LESM certificate simultaneously.

6 Experimental Evaluation

We implemented a prototype of our method which we call SuperDP [1], standing for **Super**martingale-based **D**ifferential **P**rivacy refutation. Our prototype tool takes as input (i) a probabilistic program which is translated into a pCFG C, (ii) a similarity relation Φ, and (iii) a privacy budget $\epsilon \geq 0$. It then runs the algorithm in Section 5 to refute ϵ-DP of C. If successful, the tool returns an expectation super/sub-martingale witness $(\mathbf{x}_1, \mathbf{x}_2, f, U_f, L_f)$ that refutes ϵ-DP of C.

Research questions. The goal of our experiments is to answer three research questions (RQs):

(RQ1) *Effectiveness.* Is SuperDP effective in refuting ϵ-DP of stochastic mechanisms, including those that are out of the reach of existing methods?

(RQ2) *Efficiency.* Is SuperDP able to refute ϵ-DP at runtimes competitive with the state of the art?

(RQ3) *Limitations.* What are the practical limitations of our approach?

6.1 Experimental Setup

Implementation. SuperDP is implemented in Java and uses Z3 [28] and MathSAT5 [25] as backend SMT-solvers to solve the system of polynomial constraints obtained in Step 4 of our algorithm in Section 5. It also uses ASPIC [35] to generate supporting linear invariants. Given an input as stated above, SuperDP runs the algorithm in Section 5 with maximal polynomial degrees $D = 1, \ldots, 6$. We note that our algorithm requires the number of program variables to be finite and statically bounded. Hence, if the input PP contains an array whose size is an input parameter (as do the first 7 benchmarks in Table 2), we start the DP refutation analysis with array size 2. We then iteratively increment the array size by 1 and run our algorithm in Section 5 until we reach an array size for which SuperDP successfully refutes ϵ-DP, or until timeout is reached (5 minutes per benchmark).

Baselines. We compare the performance of our prototype against two state of the art tools for reasoning about DP that support ϵ-DP refutation:

- *Static analysis baseline.* CheckDP [59] is the only available static analysis tool that is fully automated and supports ϵ-DP refutation, hence we use it as the first baseline. CheckDP tries to find an alignment between random variables along executions on a pair of similar inputs. If an alignment is found, the program is guaranteed to satisfy ϵ-DP; otherwise, CheckDP refutes ϵ-DP by computing a triple $(\mathbf{x}^1_{in}, \mathbf{x}^2_{in}, A)$ with $\mathbb{P}_{\mathbf{x}^1_{in}}[Output(A)] > e^{\epsilon} \cdot \mathbb{P}_{\mathbf{x}^2_{in}}[Output(A)]$.
- *Dynamic analysis baseline.* StatDP [30] is a statistical framework for DP refutation. It carefully searches for a pair of similar inputs that might violate the DP property, runs the program many times on those inputs, and evaluates them by hypothesis testing and p-value computation. Smaller p-values indicate stronger evidence against the DP claim. In what follows, we assume that StatDP refutes ϵ-DP if the p-value it returns is less than 0.1.

Other tools. We also attempted to run DiPC [6] as a baseline, but it always terminated with runtime errors, so we are unable to report results for it. We note, however, that only three of our benchmarks (`RR-1`,`RR-2`, and `lowprob`) are compatible with their supported syntax. DP-Sniper [10] and DP-Finder [9] are statistical methods that require huge computational resources (they use 500 GB of memory and 128 CPU cores in their experiments), hence we omit them from our experiments.

6.2 Benchmarks and Setup

All experiments were run on an Ubuntu 24.04 machine with an 11th Gen Intel Core i5 CPU and 16 GB RAM with a timeout of 5 minutes. Our prototype and CheckDP use only one core of the CPU, whereas StatDP was allowed to use up to 8 cores. For benchmarks, we consider 15 probabilistic programs collected from the DP literature and run our prototype and the two baselines on them:

[1] Available at https://github.com/ChatterjeeGroup-ISTA/SuperDP and https://doi.org/10.5281/zenodo.18930113

- The first 9 benchmarks are standard stochastic mechanisms from the differential privacy literature [33] (first 9 rows in Table 2, where SVT stands for "Sparse Vector Technique" and RR stands for "Randomized Response"). BadSmartSum is an incorrect variant of SmartSum introduced in [59]. The parameter M in both SmartSum and BadSmartSum benchmarks is set to 3. We consider two versions of the randomized response mechanism, RR-1 is shown in Fig. 1a and RR-2 is shown in the extended version [21].
- The geometric mechanism (detailed in the extended version [21]) first samples the noise variable z from a geometric distribution (lines l_2 to l_6) and *sign* from a Bernoulli and adds $(-1)^{sign} \cdot z$ to the input. This mechanism is particularly interesting because the geometric sampling is done by an almost-sure terminating loop that is not statically bounded.
- The PrivBernoulli mechanism was considered in LightDP [63] and its details are shown in the extended version [21].Intuitively, given and input x, it returns the result of an x-biased coin toss. We consider two variants of this mechanism: (i) a variant where the input x can take any value in the range $[0, 1]$, which provides no DP guarantees, and (ii) a variant where x is limited to $[\frac{1}{3}, \frac{2}{3}]$, which satisfies $(\ln 2)$-DP. CheckDP does not support these benchmarks, since all randomness in their input has to come from Laplacian samplings.
- The lowprob benchmark is motivated by the StatDP paper [30] (details available in the extended version [21]). Given any input, this mechanism returns 0 with probability at least $1 - 10^{-6}$. This means that a statistical sampling-based method requires many samples with $x = 1$ as input in order to observe 1 on output. Without such an observation, it cannot differentiate between lowprob and a mechanism that always returns 0 and satisfies 0-DP.
- RE stands for "Random element" which is a toy example that, given an array of length 5, returns one of the elements of the array uniformly at random, hence trivially violating privacy. It satisfies (ϵ, δ)-DP for $\epsilon = 0$ and $\delta = 0.2$, but not ϵ-DP for any value of ϵ.
- The Uniform noise mechanism [37] is similar to the histogram mechanism, but instead of adding Laplaccian noise, it adds $Uniform_{[-0.1,0.1]}$ noise to the elements of the input array.

Among our benchmarks, the geometric benchmark satisfies condition (C4) of Definition 4.3, while all others satisfy condition (C1) of OST-soundness (statically bounded termination).

6.3 Results

Table 2 summarizes our experimental results. Green cells indicate the best performing tools on each benchmark, e.g. on the PartialSum benchmark, all the tools could refute up to 0.9-DP and CheckDP was the fastest taking 0.7s. We discuss our results by considering each of the research questions (RQ1-3) separately.

(RQ1) The first question is concerned with the effectiveness of our method for refuting ϵ-DP. We argue that the answer to this question is positive by analyzing the Refuted DP columns in Table 2. It can be seen that SuperDP refutes ϵ-DP with the largest value of ϵ compared to the baselines on 13/15 benchmarks (the 2 failed instances are discussed under RQ3 below). Specifically, on the 9 benchmarks that satisfy ϵ-DP for some finite value of ϵ, SuperDP is able

[1]The syntax of CheckDP only supports sampling instructions from Laplacian distributions, but not from e.g. Gaussian, Uniform, or Bernoulli distributions. Hence, some of our benchmarks are not supported by CheckDP. To make CheckDP applicable to benchmarks with probabilistic branching, when providing inputs to it, we replace every instruction of the form "`if prob(c)`" with "tmp `:= Lap(1); if` $tmp \geq t$" for an appropriate value of t such that $\Pr[Lap(1) \geq t] \simeq c$.

[2]On the randomized response benchmarks (RR-1 and RR-2), SuperDP and StatDP refute ϵ-DP by computing correct and tight bounds on ϵ. However, the result of CheckDP on these two benchmarks is unsound as the reported value of ϵ for which ϵ-DP is refuted exceeds the theoretically known value of ϵ for which the mechanism is ϵ-DP. We report this as a subtle bug in their implementation of counterexample validation, which we believe to be due to these examples not admitting any random variable alignment hence CheckDP refutes ϵ-DP for any value of ϵ.

Table 2. Summary of our experimental results. The "DP" column shows the theoretically proved smallest value of ϵ for which a mechanism is ϵ-DP, where ∞ means that the benchmark is not ϵ-DP for any ϵ. The "Refuted DP" columns show the largest ϵ for which ϵ-DP was successfully refuted by each tool (larger is better), the "Time" columns show the runtime of each tool in seconds, and the $|M^D|$ column indicates the size of the template used in SuperDP. Green cells indicate the tool with the best performance on each benchmark. "NS" stands for "Not Supported"[1], "TO" for "Timeout", and "F" for "Fail". Grey cells indicate some incorrect answers by the baseline tools that we observed in our experiments.[2]

Benchmark	DP	SuperDP			CheckDP [59]		StatDP [30]	
		Refuted DP	Time	$\lvert M^D\rvert$	Refuted DP	Time	Refuted DP	Time
PartialSum	1	0.9	1.3	70	0.9	0.7	0.9	3.4
Histogram	1	0.9	0.3	35	0.9	0.4	TO	TO
SmartSum	2	1.9	131.1	210	1.9	1.5	TO	TO
BadSmartSum	∞	>15	0.4	28	>15	0.9	TO	TO
NoisyMax	1	TO	TO	-	0.9	13.7	0.9	12.4
SVT	1	TO	TO	-	0.9	60.1	0.9	52
Gaussian	∞	>15	0.9	15	NS	NS	TO	TO
RR-1	$\ln 3$	1	0.7	3	>15	0.9	1	1.9
RR-2	$\ln 1.5$	0.4	0.4	3	>15	0.8	0.4	1.8
geometric	$\ln 2$	0.6	0.7	15	F	F	1.3	17.8
PrivBernoulli-1	∞	>15	0.1	3	NS	NS	11.5	1.9
PrivBernoulli-2	$\ln 2$	0.6	0.1	3	NS	NS	0	12.1
lowprob	∞	>15	0.1	3	F	F	F	F
RE	∞	>15	0.1	28	>15	1.2	F	F
Uniform noise	∞	>15	2.2	84	NS	NS	F	F

to refute ϵ-DP with a 0.1-tight value of ϵ for 7 instances, while CheckDP and StatDP refute such tight bounds for only 5 instances. On the remaining 6 benchmarks that do not satisfy ϵ-DP for any finite value of ϵ, SuperDP successfully refutes up to 15-DP for all instances, while CheckDP can prove similar bounds for only 2 and StatDP for none.

We note that the result of StatDP on the `geometric` benchmark is unsound. We believe this is due to the statistical nature of the method, hence the tool can output false-positive and false-negative results with small but non-negligible probability. Moreover, on the `lowprob` benchmark, StatDP cannot refute ϵ-DP for any ϵ since it requires many samples to observe a single non-zero output. On the other hand, our SuperDP does not suffer from this limitation and is able to refute ϵ-DP tightly. We also note that CheckDP exited with a runtime error in two instances (`geometric` and `lowprob`) without proving or disproving DP.

(RQ2) The second question asks about the efficiency of our approach. By analyzing the `Time` columns in Table 2, we argue that the answer to this question is also positive, i.e., SuperDP's runtime is competitive, and in many cases better, than the baselines. Specifically, SuperDP outperforms the baselines in terms of runtime in 11/15 cases. On the `SmartSum` benchmark, our approach takes more than 2 minutes to terminate because the benchmark has more pCFG locations (larger $|L|$) compared to `PartialSum` and `Histogram`, which results in larger template sizes and a larger system of constraints in the last step of our algorithm.

(RQ3) The third question is concerned with understanding practical limitations of our approach. Based on the results presented in Table 2, we observe the following limitations:

– *Symbolic reasoning about probabilities of* `if`*-branching.* Our tool timeouts when solving `NoisyMax` and `SVT` benchmarks, while both baselines can refute up to 0.9-DP. This is because both benchmarks contain code fragments that look like

$$x := Lap(1);$$
$$\texttt{if}(x \geq y) \quad \ldots$$

However, reasoning about such `if`-statements by using polynomial templates is difficult, since the probability of $(x \geq y)$ needs to be symbolically computed. Hence, our method sometimes struggles with examples that require explicit reasoning about probabilities of conditional branching that cannot be captured by polynomial templates.

– *Large systems of constraints.* As mentioned above, in cases like `SmartSum`, our approach needs to solve large systems of constraints which may be computationally expensive for SMT-solvers. Moreover, in the `Uniform noise` benchmark, our approach requires degree 6 polynomial templates to refute 15-DP. Using high-degree polynomial templates results in larger systems of constraints, which may be harder for SMT-solvers to solve.

Summary. As demonstrated by our results in Table 2, SuperDP can successfully refute ϵ-DP for 13/15 benchmarks, at runtimes that are consistently lower than 3 seconds in all but one case. This clearly positively answers our first two research questions (RQ1) and (RQ2). We also discussed two practical limitations of our approach in answer to (RQ3). The first limitation arises in examples that require symbolic reasoning about probabilities of `if`-branching which may be a challenge when using polynomial templates, while the second limitation is due to instances where large systems of constraints need to be solved.

7 Related Work

Automated analysis of DP has seen a lot of interest in the verification literature. Below we discuss the literature related to refutation of DP. In the extended version [21], we discuss the works on verification of DP properties as well. We classify existing methods for DP refutation into (i) static analysis methods (including our work), and (ii) dynamic (sampling-based) methods.

Static analysis methods. These methods try to refute DP by statically analyzing the program rather than executing it. DiPC [6] considers programs with finite inputs and outputs and does not allow assignment to real or integer variables inside a while loop. They model the given program M as a finite-state discrete-time Markov chain (DTMC) and exhaustively search for similar inputs a, a', and an output b such that $\Pr[M(a) = b] > e^{\epsilon} \Pr[M(a') = b]$. Due to the restriction to finite domain inputs and outputs, they have to consider discretizations of programs when they contain infinite-ranges of inputs or outputs. In [34], the input program, which is limited to have countable inputs and outputs, is executed in a relational and symbolic way. After the execution, the method checks if there exists a coupling between the generated outputs. They also introduce several strategies to look for counterexamples for disproving DP in case their method fails to find a coupling. In contrast, our method does not impose any restrictions on the domains of program variables or places where assignments can occur in the program.

In a separate line of work, CheckDP [59] is the successor of LightDP [63] and ShadowDP [60], which is fully automated with a verify-invalidate loop that looks for randomness alignments for proving DP. If in any loop iteration, the proposed randomness alignment is invalidated by a counterexample that cannot be fixed by a new alignment, the counterexample is passed to PSI [36] to make sure it is a valid counterexample for the claimed DP bound. The main limitations of CheckDP compared to our approach are: (i) the only source of randomness in their syntax is sampling from the Laplace distribution, (ii) they provide no completeness guarantees, and (iii) we encountered a subtle bug in their implementation of counterexample validation using PSI. In contrast, SuperDP provides soundness and conditional semi-completeness guarantees as explained in Section 4 and Section 5, and supports general probability distributions including Laplace, Gaussian, and Bernoulli.

Dynamic analysis methods. In contrast to static analysis approaches that refute DP without running the input program, statistical methods execute the program multiple times and try to infer/refute DP based on the samples that they observe. StatDP [30] is one such method that

considers input pairs satisfying specific patterns and runs the program many times on them. It then conducts hypothesis testing on the observed samples and computes a p-value, which is a probabilistic estimate of how likely it is that the program satisfies the claimed DP property. A low p-value indicates that the input program most likely does not satisfy the claimed DP. DP-Finder [9] uses samples taken from the program to approximate the function $\epsilon(x, x', A)$ as the amount of privacy loss corresponding to similar inputs x, x' and output set A. They then find a pair (x, x') that maximizes the approximated privacy loss function and lastly use PSI to compute the exact value of $\epsilon(x, x', A)$. DP-Sniper [10] uses classifiers (e.g. logistic regression or neural networks) to compute approximations of $\epsilon(x, x', A)$. Although these methods only require a black-box access to the input program, which is an advantage, they suffer from the following limitations: (i) they are all based on approximations and do not provide formal soundness guarantees (e.g. StatDP fails on `lowprob` and `geometric` benchmarks in our experiments), (ii) they might require many samples in order to make their computations more accurate which requires huge computing power (e.g. DP-Finder's experiments were conducted with 500 GB RAM and 128 CPU cores). Our approach, in contrast, requires white-box access to the program and provides formal soundness and semi-completeness guarantees, while requiring modest computational resources.

Martingale-based probabilistic program analysis. Our method leverages upper expectation supermartingales and lower expectation submartingales of [19, 20] to formally and automatically reason about DP. Supermartingale and submartingale processes from probability theory [61] have been extensively used to design fully automated approaches for probabilistic program and model analysis, for properties such as termination and reachability [1, 3, 13, 15, 16, 18, 24, 44, 45, 47, 53], safety [23, 46, 52, 58, 64], reach-avoidance [62, 66], cost analysis [17, 49, 57], sensitivity [56], and more recently general omega-regular properties [2]. Similarly to our method, many works automate computation of supermartingale and submartingale witnesses by following a template-based synthesis approach. In particular, our algorithm is also related to the work [65] on differential cost analysis in non-probabilistic programs, where the difference in cost usage is analyzed by simultaneously computing polynomial bounds on cost usage in two programs. To the best of our knowledge, our work is the first to use supermartingale and submartingale witnesses, as well as simultaneous reasoning about two expectation bounds of a function, towards analyzing DP.

8 Conclusion

We discuss the key contributions of our method along with its limitations. We present a novel method for ϵ-DP refutation in probabilistic programs by reasoning about expectation mismatch of a non-negative function over program outputs. We introduce a sound and complete proof rule for ϵ-DP refutation based on expectation supermartingale and submartingale witnesses. Our approach reduces the ϵ-DP refutation problem to finding a non-negative function over outputs with significant expected value difference on two similar inputs, circumventing the need to directly reason about the probability of output events. This enables us to design a fully automated, sound and semi-complete algorithm for synthesizing witnesses for ϵ-DP refutation.

In summary, the key theoretical novelty is to ensure all four desirable properties discussed in Section 1. Moreover, our experiments demonstrate that SuperDP can effectively refute ϵ-DP for a wide range of challenging examples, including those that were beyond the reach of previous methods. This shows the practical potential of our approach. The limitations of our work can be summarized as (i) our approach for ϵ-DP refutation is limited to polynomial arithmetic programs and its extension to other classes of programs remains open; and (ii) our approach is limited to ϵ-DP refutation and its extension to other notions of privacy such as (ϵ, δ)-DP refutation and to DP verification is another open question. Addressing these are interesting directions of future work.

Data Availability Statement

The artifact supporting the findings of this study, which includes the underlying datasets, software code, and experiments, is publicly available in Zenodo [22].

Acknowledgments

The authors would like to thank Petr Novotný for valuable discussions that helped shape this work. This research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant (Proposal ID: 25-SIS-SMU-009), Vienna Science and Technology Fund (WWTF), State of Lower Austria [Grant ID 10.47379/ICT25017], ERC CoG 863818 (ForM-SMArt), and Austrian Science Fund (FWF) 10.55776/COE12.

References

[1] Alessandro Abate, Mirco Giacobbe, and Diptarko Roy. 2021. Learning Probabilistic Termination Proofs. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12760)*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer, 3–26. doi:10.1007/978-3-030-81688-9_1

[2] Alessandro Abate, Mirco Giacobbe, and Diptarko Roy. 2024. Stochastic Omega-Regular Verification and Control with Supermartingales. In *CAV (3) (Lecture Notes in Computer Science, Vol. 14683)*. Springer, 395–419. doi:10.1007/978-3-031-65633-0_18

[3] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018). doi:10.1145/3158122

[4] Ali Asadi, Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Mohammad Mahdavi. 2021. Polynomial reachability witnesses via Stellensätze. In *PLDI*. ACM, 772–787. doi:10.1145/3453483.3454076

[5] Martin Avanzini, Gilles Barthe, Davide Davoli, and Benjamin Grégoire. 2025. A Quantitative Probabilistic Relational Hoare Logic. *Proc. ACM Program. Lang.* 9, POPL (2025), 1167–1195. doi:10.1145/3704876

[6] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2020. Deciding Differential Privacy for Programs with Finite Inputs and Outputs. In *LICS*. ACM, 141–154. doi:10.1145/3373718.3394796

[7] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Proving Differential Privacy via Probabilistic Couplings. In *LICS*. ACM, 749–758. doi:10.1145/2933575.2934554

[8] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella-Béguelin. 2013. Probabilistic Relational Reasoning for Differential Privacy. *ACM Trans. Program. Lang. Syst.* 35, 3 (2013), 9:1–9:49. doi:10.1145/2492061

[9] Benjamin Bichsel, Timon Gehr, Dana Drachsler-Cohen, Petar Tsankov, and Martin T. Vechev. 2018. DP-Finder: Finding Differential Privacy Violations by Sampling and Optimization. In *CCS*. ACM, 508–524. doi:10.1145/3243734.3243863

[10] Benjamin Bichsel, Samuel Steffen, Ilija Bogunovic, and Martin T. Vechev. 2021. DP-Sniper: Black-Box Discovery of Differential Privacy Violations using Classifiers. In *SP*. IEEE, 391–409. doi:10.1109/SP40001.2021.00081

[11] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 441–459. doi:10.1145/3132747.3132769

[12] Mark Bun, Marco Gaboardi, and Ludmila Glinskih. 2022. The Complexity of Verifying Boolean Programs as Differentially Private. In *CSF*. IEEE, 396–411. doi:10.1109/CSF54842.2022.9919653

[13] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 511–526. doi:10.1007/978-3-642-39799-8_34

[14] Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Ehsan Kafshdar Goharshady. 2020. Polynomial invariant generation for non-deterministic recursive programs *(PLDI 2020)*. doi:10.1145/3385412.3385969

[15] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 40, 2 (2018). doi:10.1145/3174800

[16] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Dorde Zikelic. 2022. Sound and Complete Certificates for Quantitative Termination Analysis of Probabilistic Programs. In *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13371)*, Sharon Shoham and Yakir Vizel (Eds.). Springer, 55–78. doi:10.1007/978-3-031-13185-1_4

[17] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Dorde Zikelic. 2024. Quantitative Bounds on Resource Usage of Probabilistic Programs. *Proc. ACM Program. Lang.* 8, OOPSLA1 (2024), 362–391. doi:10.1145/3649824

[18] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, Jiri Zárevúcky, and Dorde Zikelic. 2023. On Lexicographic Proof Rules for Probabilistic Termination. *Formal Aspects Comput.* 35, 2 (2023), 11:1–11:25. doi:10.1145/3585391

[19] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, and Dorde Zikelic. 2024. Equivalence and Similarity Refutation for Probabilistic Programs. *Proc. ACM Program. Lang.* 8, PLDI (2024), 2098–2122. doi:10.1145/3656462

[20] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, and Dorde Zikelic. 2025. Refuting Equivalence in Probabilistic Programs with Conditioning. In *TACAS (2) (Lecture Notes in Computer Science, Vol. 15697)*. Springer, 279–300. doi:10.1007/978-3-031-90653-4_14

[21] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, and Đorđe Žikelić. 2026. SuperDP: Differential Privacy Refutation via Supermartingales. *arXiv preprint arXiv:2603.26215* (2026).

[22] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, and Đorđe Žikelić. 2026. *SuperDP: Differential Privacy Refutation via Supermartingales*. doi:10.5281/zenodo.19399862

[23] Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. 2017. Stochastic invariants for probabilistic termination. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 145–160. doi:10.1145/3009837.3009873

[24] Jianhui Chen and Fei He. 2020. Proving almost-sure termination by omega-regular decomposition. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 869–882. doi:10.1145/3385412.3386002

[25] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. 2013. The MathSAT5 SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*. doi:10.1007/978-3-642-36742-7_7

[26] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. 2003. Linear Invariant Generation Using Non-linear Constraint Solving. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2725)*, Warren A. Hunt Jr. and Fabio Somenzi (Eds.). Springer, 420–432. doi:10.1007/978-3-540-45069-6_39

[27] Michael Colón and Henny Sipma. 2001. Synthesis of Linear Ranking Functions. In *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2031)*, Tiziana Margaria and Wang Yi (Eds.). Springer, 67–81. doi:10.1007/3-540-45319-9_6

[28] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*. doi:10.1007/978-3-540-78800-3_24

[29] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 3571–3580. https://proceedings.neurips.cc/paper/2017/hash/253614bbac999b38b5b60cae531c4969-Abstract.html

[30] Zeyu Ding, Yuxin Wang, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. 2018. Detecting Violations of Differential Privacy. In *CCS*. ACM, 475–489. doi:10.1145/3243734.3243818

[31] Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 4052)*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.). Springer, 1–12. doi:10.1007/11787006_1

[32] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan. 2009. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, Michael Mitzenmacher (Ed.). ACM, 381–390. doi:10.1145/1536414.1536467

[33] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407. doi:10.1561/0400000042

[34] Gian Pietro Farina, Stephen Chong, and Marco Gaboardi. 2021. Coupled Relational Symbolic Execution for Differential Privacy. In *ESOP (Lecture Notes in Computer Science, Vol. 12648)*. Springer, 207–233. doi:10.1007/978-3-030-72019-3_8

[35] Paul Feautrier and Laure Gonnord. [n. d.]. Accelerated Invariant Generation for C Programs with Aspic and C2fsm. *Electronic Notes in Theoretical Computer Science* ([n. d.]). doi:10.1016/j.entcs.2010.09.014

[36] Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *CAV (1) (Lecture Notes in Computer Science, Vol. 9779)*. Springer, 62–83. doi:10.1007/978-3-319-41528-4_4

[37] Quan Geng and Pramod Viswanath. 2016. The Optimal Noise-Adding Mechanism in Differential Privacy. *IEEE Trans. Inf. Theory* 62, 2 (2016), 925–951. doi:10.1109/TIT.2015.2504967

[38] Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nat.* 521, 7553 (2015), 452–459. doi:10.1038/NATURE14541

[39] Ji Guan, Wang Fang, Mingyu Huang, and Mingsheng Ying. 2023. Detecting Violations of Differential Privacy for Quantum Algorithms. In *CCS*. ACM, 2277–2291. doi:10.1145/3576915.3623108

[40] David Handelman. 1988. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific J. Math.* 132, 1 (1988), 35 – 62.

[41] Noah M. Johnson, Joseph P. Near, and Dawn Song. 2018. Towards Practical Differential Privacy for SQL Queries. *Proc. VLDB Endow.* 11, 5 (2018), 526–539. doi:10.1145/3187009.3177733

[42] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5 (2018), 30:1–30:68. doi:10.1145/3208102

[43] Min Lyu, Dong Su, and Ninghui Li. 2017. Understanding the Sparse Vector Technique for Differential Privacy. *Proc. VLDB Endow.* 10, 6 (2017), 637–648. doi:10.14778/3055330.3055331

[44] Rupak Majumdar and V. R. Sathiyanarayana. 2024. Positive Almost-Sure Termination: Complexity and Proof Rules. *Proc. ACM Program. Lang.* 8, POPL (2024), 1089–1117. doi:10.1145/3632879

[45] Rupak Majumdar and V. R. Sathiyanarayana. 2025. Sound and Complete Proof Rules for Probabilistic Termination. *Proc. ACM Program. Lang.* 9, POPL (2025), 1871–1902. doi:10.1145/3704899

[46] Frederik Baymler Mathiesen, Simeon C. Calvert, and Luca Laurenti. 2023. Safety Certification for Stochastic Systems via Neural Barrier Functions. *IEEE Control. Syst. Lett.* 7 (2023), 973–978. doi:10.1109/LCSYS.2022.3229865

[47] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2018. A new proof rule for almost-sure termination. *Proc. ACM Program. Lang.* 2, POPL (2018), 33:1–33:28. doi:10.1145/3158121

[48] Sean P Meyn and Richard L Tweedie. 2012. *Markov chains and stochastic stability*. Springer Science & Business Media. doi:10.1007/978-1-4471-3267-7

[49] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: resource analysis for probabilistic programs. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, Jeffrey S. Foster and Dan Grossman (Eds.). ACM, 496–512. doi:10.1145/3192366.3192394

[50] Andreas Podelski and Andrey Rybalchenko. 2004. A Complete Method for the Synthesis of Linear Ranking Functions. In *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, Italy, January 11-13, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 2937)*, Bernhard Steffen and Giorgio Levi (Eds.). Springer, 239–251. doi:10.1007/978-3-540-24622-0_20

[51] Andreas Podelski and Andrey Rybalchenko. 2004. Transition Invariants. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*. IEEE Computer Society, 32–41. doi:10.1109/LICS.2004.1319598

[52] Stephen Prajna, Ali Jadbabaie, and George J. Pappas. 2007. A Framework for Worst-Case and Stochastic Safety Verification Using Barrier Certificates. *IEEE Trans. Autom. Control.* 52, 8 (2007), 1415–1428. doi:10.1109/TAC.2007.902736

[53] Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. 2021. Ranking and Repulsing Supermartingales for Reachability in Randomized Programs. *ACM Trans. Program. Lang. Syst.* 43, 2 (2021), 5:1–5:46. doi:10.1145/3450967

[54] The OpenDP Team. 2021. The OpenDP Library. https://opendp.org

[55] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *CoRR* abs/1809.10756 (2018). arXiv:1809.10756 http://arxiv.org/abs/1809.10756

[56] Peixin Wang, Hongfei Fu, Krishnendu Chatterjee, Yuxin Deng, and Ming Xu. 2020. Proving expected sensitivity of probabilistic programs with randomized variable-dependent termination time. *Proc. ACM Program. Lang.* 4, POPL (2020), 25:1–25:30. doi:10.1145/3371093

[57] Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost analysis of nondeterministic probabilistic programs. In *PLDI*. doi:10.1145/3314221.3314581

[58] Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, and C.-H. Luke Ong. 2024. Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving. *Proc. ACM Program. Lang.* 8, PLDI (2024), 1361–1386. doi:10.1145/3656432

[59] Yuxin Wang, Zeyu Ding, Daniel Kifer, and Danfeng Zhang. 2020. CheckDP: An Automated and Integrated Approach for Proving Differential Privacy or Finding Precise Counterexamples. In *CCS*. ACM, 919–938. doi:10.1145/3372297.3417282

[60] Yuxin Wang, Zeyu Ding, Guanhong Wang, Daniel Kifer, and Danfeng Zhang. 2019. Proving differential privacy with shadow execution. In *PLDI*. ACM, 655–669. doi:10.1145/3314221.3314619

[61] David Williams. 1991. *Probability with Martingales*. Cambridge University Press.

[62] Bai Xue, Renjue Li, Naijun Zhan, and Martin Fränzle. 2021. Reach-avoid Analysis for Stochastic Discrete-time Systems. In *2021 American Control Conference, ACC 2021, New Orleans, LA, USA, May 25-28, 2021*. IEEE, 4879–4885.

doi:10.23919/ACC50511.2021.9483095

[63] Danfeng Zhang and Daniel Kifer. 2017. LightDP: towards automating differential privacy proofs. In *POPL*. ACM, 888–901. doi:10.1145/3009837.3009884

[64] Dapeng Zhi, Peixin Wang, Si Liu, C.-H. Luke Ong, and Min Zhang. 2024. Unifying Qualitative and Quantitative Safety Verification of DNN-Controlled Systems. In *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14682)*, Arie Gurfinkel and Vijay Ganesh (Eds.). Springer, 401–426. doi:10.1007/978-3-031-65630-9_20

[65] Dorde Zikelic, Bor-Yuh Evan Chang, Pauline Bolignano, and Franco Raimondi. 2022. Differential cost analysis with simultaneous potentials and anti-potentials. In *PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022*, Ranjit Jhala and Isil Dillig (Eds.). ACM, 442–457. doi:10.1145/3519939.3523435

[66] Dorde Zikelic, Mathias Lechner, Thomas A. Henzinger, and Krishnendu Chatterjee. 2023. Learning Control Policies for Stochastic Systems with Reach-Avoid Guarantees. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 11926–11935. doi:10.1609/AAAI.V37I10.26407

Received 2025-11-12; accepted 2026-04-03