Deciding Differential Privacy for Programs with Finite Inputs and Outputs

Gilles Barthe
MPI Security and Privacy and IMDEA Software Institute
Germany
gjbarthe@gmail.com

Rohit Chadha
University of Missouri
USA
chadhar@missouri.edu

Vishal Jagannath
University of Illinois at Urbana-Campaign
USA
vishalrjagan@gmail.com

A. Prasad Sistla
University of Illinois at Chicago
USA
sistla@uic.edu

Mahesh Viswanathan
University of Illinois at Urbana-Campaign
USA
vmahesh@illinois.edu

Abstract

Differential privacy is a *de facto* standard for statistical computations over databases that contain private data. Its main and rather surprising strength is to guarantee individual privacy and yet allow for accurate statistical results. Thanks to its mathematical definition, differential privacy is also a natural target for formal analysis. A broad line of work develops and uses logical methods for proving privacy. A more recent and complementary line of work uses statistical methods for finding privacy violations. Although both lines of work are practically successful, they elide the fundamental question of decidability.

This paper studies the decidability of differential privacy. We first establish that checking differential privacy is undecidable even if one restricts to programs having a single Boolean input and a single Boolean output. Then, we define a non-trivial class of programs and provide a decision procedure for checking the differential privacy of a program in this class. Our procedure takes as input a program P parametrized by a privacy budget ϵ and either establishes the differential privacy for all possible values of ϵ or generates a counter-example. In addition, our procedure works for both to ϵ-differential privacy and (ϵ, δ)-differential privacy. Technically, the decision procedure is based on a novel and judicious encoding of the semantics of programs in our class into a decidable fragment of the first-order theory of the reals with exponentiation. We implement our procedure and use it for (dis)proving privacy bounds for many well-known examples, including randomized response, histogram, report noisy max and sparse vector.

CCS Concepts: • **Security and privacy → Logic and verification**.

Keywords: differential privacy; decision procedure; sparse vector

ACM Reference Format:
Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2020. Deciding Differential Privacy for Programs with Finite Inputs and Outputs. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3373718.3394796

1 Introduction

Differential privacy [18] is a gold standard for the privacy of statistical computations. Differential privacy ensures that running the algorithm on any two "adjacent" databases yields two "approximately" equal distributions, where two databases are adjacent if they differ in a single element, and two distributions are approximately equivalent if their distance is small w.r.t. some metric specified by privacy parameter ϵ and error parameter δ. Thus, differential privacy delivers a very strong form of individual privacy. Yet, and somewhat surprisingly, it is possible to develop differentially private algorithms for many tasks. Moreover, the algorithms are useful, in the sense that their results have reasonable accuracy. However, designing differentially private algorithms is difficult, and the privacy analysis can be error-prone, as witnessed by the example of the sparse vector technique.

This difficulty has motivated the development of formal approaches for analyzing differentially private algorithms (see [7] for a survey and the related work section of this

LICS '20, July 8–11, 2020, Saarbrücken, Germany

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394796

paper). Broadly, two successful lines of work have emerged. The first line of work develops sound proof systems to establish differential privacy and uses these proof systems to prove the privacy of well-known and intricate examples [1, 5, 6, 8, 16, 21, 31, 33, 34]. The second line of work searches for counter-examples to demonstrate the violation of differential privacy [9, 17]. Unfortunately, both lines of work elide the question of decidability. As previous experience in formal verification suggests, understanding decidable fragments of a problem not only help advance our theoretical knowledge, but can form the basis of practical tools when combined with ideas like abstraction and composition.

The goal of this paper is, therefore, to study the decision problem for differential privacy, and to make a first attempt at delineating the decidability/undecidability boundary. As a first contribution, we show that, as expected, checking differential privacy is computationally undecidable. Our undecidability result holds even if one restricts to programs having a single Boolean input and a single Boolean output. Given the undecidability result, we then consider the task of identifying a rich class of programs, that encompasses many known examples, for which checking differential privacy nonetheless is decidable. We impose two desiderata:

1. the class of programs must include programs with real-valued variables, and more generally, with variables over infinite domains. This requirement is critical for the method to cover a broad class of differential privacy algorithms;
2. the programs themselves are parametrized by the privacy parameter ϵ (throughout the paper, we assume that the error parameter δ is a function of ϵ), and the decision procedure should decide privacy for all possible instances of the privacy parameter ϵ. This requirement is motivated by the fact, supported by practice, that differential privacy algorithms are typically parametrized by ϵ, and well-designed algorithms are private not only for a single value of ϵ, but typically for all positive values of ϵ.

We focus our attention on programs whose input and output spaces are finite. Note that such programs need not be finite-state, as per our first requirement, they could use program variables ranging over infinite (even uncountable) domains to carry out the computation. We introduce a class of programs, called DiPWhile, which are probabilistic while programs, for which the problem of checking differential privacy is decidable. We succeed in carefully balancing decidability and expressivity, by judiciously delineating the use of real-valued and integer-valued variables. Intuitively, the main restriction we impose is that these infinite-valued variables be used only to directly influence the program control-flow and not the data-flow that leads to the computation of the final output. More precisely, in an execution, the program output value depends only on the input, values sampled from user-defined distributions and the exponential mechanism, and the branch conditions on the control flow path taken. The sampled values of real/integer variables affect only the branch conditions. Thus, the output values depend only on the branch conditions satisfied by the sampled real/integer variable values, but not on their actual sampled values. This restriction, though severe, turns out to capture many prominent differential privacy algorithms, including Report Noisy Max and Sparse Vector Technique (see Section 8 on experiments).

Key observations that enable us to establish decidability of DiPWhile programs are as follows. The first result is that the semantics of DiPWhile-programs can be defined using parametrized, *finite-state* Markov chains [1]. The fact that the semantics is definable using only finitely-many states is a surprising observation because our programs have both integer and real-valued variables, and hence a naïve semantics yields uncountably many possible states. Our crucial insight here is that a precise semantics for DiPWhile-programs is possible without tracking the explicit values of the real and integer-valued variables. Since real and integer variables are intuitively used only in influencing control-flow, the semantics only tracks the symbolic relationships between the variables. Second, we show that the transition probabilities of the Markov chain are ratios of polynomial functions in ϵ and e^{ϵ}, where e is the Euler's constant; this was a difficult result to establish. These two observations together, allow us to reduce the problem of checking the differential privacy of DiPWhile-programs to the decidable fragment of the first-order theory of reals with exponentials, identified by McCallum and Weispfenning [28].

We leverage our decision procedure to build a stand-alone tool for checking ϵ- or $(\epsilon, \delta(\epsilon))$-differential privacy of mechanisms specified by DiPWhile-programs, for all values of ϵ. We have implemented our decision procedure in a tool that we call DiPC (**Di**fferential **P**rivacy **C**hecker). Given DiPWhile-program, our tool constructs a sentence within the McCallum-Weispfenning fragment of the theory of reals with exponentials. It then calls Mathematica® to check if the constructed sentence is true over the reals. Since our decision procedure is the *first* that can both prove differential privacy and detect its violation, we tried the tool on examples that known to be differentially private and those that are known to be not differentially private including variants of Sparse Vector, Report Noisy Max, and Histograms. DiPC successfully checked differential privacy for the former class of examples and produced counter-examples for the latter class. Our counter-examples are exact and are more compact than those discovered by prior tools.

[1] A parametrized Markov chain is a Markov chain whose transition probabilities are a function of the privacy budget.

As a contribution of independent interest, we also demonstrate how our method yields a theoretical complete under-approximation method for checking differential privacy of programs with infinite output sets. For such programs, it is possible to discretize the output domain into a finite domain, and to use the decision procedure to find privacy violations for the discretized algorithm (by post-processing, privacy violations for the discretized algorithms are also privacy violations for the original algorithm). The discretization yields a method for generating counter-examples for algorithms with infinite output sets.

We briefly contrast our results with prior work, and refer the reader to Section 9 for further details. Overall, we see our decidability results as complementary to prior works in checking differential privacy. In general, existing methods for proving or disproving differential privacy, although inherently incomplete due to the undecidability of checking differential privacy, are likely to be more efficient because they can trade-off efficiency for precision. However, the decision procedures for a sub-class of programs, like the one presented here, maybe more predictable — if a decision procedure fails to prove privacy, then it shall produce a counter-example that demonstrates that the algorithm is not differentially private. Moreover, counter-example search methods work for a fixed (ϵ) privacy parameter. As the counter-example methods are usually statistical, they may generate both false positives and false negatives. In contrast, our decision procedures work for all values for the privacy parameter and do not generate false positives or false negatives.

Contributions. We summarize our key contributions.

- We prove the undecidability of the problem of checking differential privacy of very simple programs, including those that have a single Boolean input and output. Though unsurprising, undecidability has not been previously established in any prior work.
- We prove the decidability of differential privacy for an interesting class of programs. Our method is fully automatic that can check both differential privacy *and* detect its violation by generating *counter-examples*. To the best of our knowledge, this is the first such result that encompasses sampling from integer and real-valued variables.
- We implement the decision procedure and evaluate our approach on private and non-private examples from the literature.

Due to lack of space, some proofs and other materials have been omitted. The omitted material can be located in the arXiv repository [4].

2 Primer on differential privacy

Differential privacy [18] is a rigorous definition and framework for private statistical data mining. In this model, a trusted curator with access to the database returns answers to queries made by possibly dishonest data analysts that do not have access to the database. The task of the curator is to return probabilistically noised answers, so that data analysts cannot distinguish between two databases that are adjacent, i.e. only differ in the value of a single individual. There are two common definitions: two databases are adjacent if they are exactly the same except for the presence or absence of one record, or for the difference in one record. We abstract away from any particular definition of adjacency.

Henceforth, we denote the set of real numbers, rational numbers, natural numbers and integers by $\mathbb{R}, \mathbb{Q}, \mathbb{N}$, and $\mathbb{Z}$ respectively. The Euler constant shall be denoted by e. We assume given a set $\mathcal{U}$ of inputs, and a set $\mathcal{V}$ of outputs. A randomized function P from $\mathcal{U}$ to $\mathcal{V}$ is a function that takes an input in $\mathcal{U}$ and returns a distribution over $\mathcal{V}$. For a measurable set $S \subseteq \mathcal{V}$, the probability that the output of P on u is in the set S shall be denoted by $\mathsf{Prob}(P(u) \in S)$. In the case the output set is discrete, we use $\mathsf{Prob}(P(u) = v)$ as shorthand for $\mathsf{Prob}(P(u) \in \{v\})$.

We are now ready to define differential privacy. We assume that $\mathcal{U}$ is equipped with a binary *symmetric* relation $\Phi \subseteq \mathcal{U} \times \mathcal{U}$, which we shall call the *adjacency relation*. We say that $u_1, u_2 \in \mathcal{U}$ are *adjacent* if $(u_1, u_2) \in \Phi$.

Definition 2.1. Let $\epsilon \geq 0$ and $0 \leq \delta \leq 1$. Let $\Phi \subseteq \mathcal{U} \times \mathcal{U}$ be an adjacency relation. Let P be a randomized function with inputs from $\mathcal{U}$ and outputs in $\mathcal{V}$. We say that P is (ϵ, δ)-differentially private with respect to Φ if for all measurable subsets $S \subseteq \mathcal{V}$ and $u, u' \in \mathcal{U}$ such that $(u, u') \in \Phi$,

$$\mathsf{Prob}(P(u) \in S) \leq e^{\epsilon}\, \mathsf{Prob}(P(u') \in S) + \delta$$

As usual, we say that P is ϵ-differentially private iff it is $(\epsilon, 0)$-differentially private. If the output domain is discrete, it is equivalent to require that for all $v \in \mathcal{V}$ and $u, u' \in \mathcal{U}$ such that $(u, u') \in \Phi$,

$$\mathsf{Prob}(P(u) = v) \leq e^{\epsilon}\, \mathsf{Prob}(P(u') = v)$$

Differential privacy is preserved by post-processing. Concretely, if P is an (ϵ, δ)-differentially private computation from $\mathcal{U}$ to $\mathcal{V}$, and $h : \mathcal{V} \to \mathcal{W}$ is a deterministic function, then $h \circ P$ is an (ϵ, δ)-differentially private computation from $\mathcal{U}$ to $\mathcal{W}$. In the remainder, we shall exploit post-processing to connect differential privacy of randomized computations with infinite output spaces to differential privacy of their discretizations.

Laplace Mechanism. The Laplace mechanism [18] achieves differential privacy for numerical computations by adding random noise to outputs. Given $\epsilon > 0$ and mean μ, let $\mathsf{Lap}(\epsilon, \mu)$ be the continuous distribution whose probability density function (p.d.f.) is given by

$$f_{\epsilon,\mu}(x) = \frac{\epsilon}{2}\, e^{-\epsilon|x-\mu|}.$$

$\mathsf{Lap}(\epsilon, \mu)$ is said to be the *Laplacian distribution* with mean μ and scale parameter $\frac{1}{\epsilon}$. Consider a real-valued function

$q : \mathcal{U} \to \mathbb{R}$. Assume that q is k-sensitive w.r.t. an adjacency relation Φ on $\mathcal{U}$, i.e. for every pair of adjacent values u_1 and u_2, $|q(u_1) - q(u_2)| \leq k$. Then the computation that maps u to $\mathsf{Lap}(\frac{\epsilon}{k}, q(u))$ is ϵ-differentially private.

It is sometimes convenient to consider the discrete version of the Laplace distribution. Given $\epsilon > 0$ and mean μ, let $\mathsf{DLap}(\epsilon, \mu)$ be the discrete distribution on $\mathbb{Z}$, the set of integers, whose probability mass function (p.m.f.) is

$$f_{\epsilon,\mu}(i) = \frac{1 - e^{-\epsilon}}{1 + e^{-\epsilon}}\, e^{-\epsilon|i-\mu|}.$$

$\mathsf{DLap}(\epsilon, \mu)$ is said to be the *discrete Laplacian distribution* with mean μ and scale parameter $\frac{1}{\epsilon}$. The discrete Laplace mechanism achieves the same privacy guarantees as the continuous Laplace mechanism.

Exponential mechanism. The Exponential mechanism [29] is used for making non-numerical computations private. The mechanism takes as input a value u from some input domain and a scoring function $F : \mathcal{U} \times \mathcal{V} \to \mathbb{R}$ and outputs a discrete distribution over $\mathcal{V}$. Formally, given $\epsilon > 0$ and $u \in \mathcal{U}$, the discrete distribution $\mathsf{Exp}(\epsilon, F, u)$ on $\mathcal{V}$ is given by the probability mass function:

$$h_{\epsilon,F,u}(v) = \frac{e^{\epsilon F(u,v)}}{\sum_{v \in \mathcal{V}} e^{\epsilon F(u,v)}}.$$

Suppose that the scoring function is k-sensitive w.r.t. some adjacency relation Φ on $\mathcal{U}$, i.e., for all for each pair of adjacent values u_1 and u_2 and $v \in \mathcal{V}$, $|F(u_1, r) - F(u_2, r)| \leq k$. Then the exponential mechanism is $(2k\epsilon, 0)$-differentially private w.r.t. Φ.

3 Motivating Example

Before presenting the mathematical details of our results, let us informally introduce our method by showing how it would work on an illustrative example.

Sparse Vector Technique. Several differential privacy examples require that the randomized algorithms sampling from infinite support distributions (including continuous distributions). The Sparse Vector Technique (SVT) [19, 27] was designed to answer multiple Δ-sensitive numerical queries in a differentially private fashion. The relevant information we want from queries is, which amongst them are above a threshold T. The Sparse Vector Technique as given in Algorithm 1 is designed to identify the first c queries that are above the threshold T in an ϵ-differentially private fashion.

In the program, the integer N represents the total number of queries, and the array q of length N represents the answers to queries. The array *out* represents the output array, $\perp$ represents False and $\top$ represents True. We assume that initially the constant $\perp$ is stored at each position in *out*. In the SVT technique, the $\top$ answers account for most of the privacy cost, and we can only answer c of them until we run out of the privacy budget [19, 34]. On the other hand, there is no restriction on the number of $\perp$ answers. Please observe that the SVT algorithm is parametrized by the privacy budget ϵ. Thus, the SVT algorithm can be considered as representing a class of programs, one for each $\epsilon > 0$.

Input: $q[1:N]$
Output: $out[1:N]$

$\mathsf{r}_T \leftarrow \mathsf{Lap}(\frac{\epsilon}{2\Delta}, T)$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  $\mathsf{r} \leftarrow \mathsf{Lap}(\frac{\epsilon}{4c\Delta}, q[i])$
  $\mathsf{b} \leftarrow \mathsf{r} \geq \mathsf{r}_T$
  if b **then**
    $out[i] \leftarrow \top$
    $count \leftarrow count + 1$
    if $count \geq c$ **then**
      exit
    end
  else
    $out[i] \leftarrow \perp$
  end
end

Algorithm 1: SVT algorithm (SVT1)

Given N, the input set $\mathcal{U}$ in this context is the set of N length vectors q, where the kth element $q[k]$ represents the answer to the kth query on the original database. The adjacency relation Φ on inputs is defined as follows: q_1 and q_2 are adjacent if and only if $|q_1[i] - q_2[i]| \leq 1$ for each $1 \leq i \leq N$.

Let us consider an instance of the SVT algorithm when $T = 0$, $N = 2$, $\Delta = 1$ and $c = 1$. Let us assume that all array elements in q come from the domain $\{0, 1\}$. In this case, we have four possible inputs $[0, 0]$, $[0, 1]$, $[1, 1]$, and $[1, 0]$, and three possible outputs $[\perp, \perp]$, $[\top, \perp]$, and $[\perp, \top]$.

For example, the probability of outputting $[\perp, \top]$ on input $[0, 1]$ can be computed as follows. Let X_T be a random variable with Laplacian distribution $\mathsf{Lap}(\frac{\epsilon}{2}, 0)$, X_1 be a random variable with Laplacian distribution $\mathsf{Lap}(\frac{\epsilon}{4}, 0)$ and X_2 be the random variable with Laplacian distribution $\mathsf{Lap}(\frac{\epsilon}{4}, 1)$. The probability of outputting $[\perp, \top]$ is the product of outputting of outputting $\perp$ first, which is $Prob(X_1 < X_0)$, and the conditional probability of outputting $\top$ given that $\perp$ is output, which is $Prob(X_2 \geq X_0 | X_1 < X_0)$. Note that we really require the second quantity to be conditional probability as the events $X_1 < X_0$ and $X_2 \geq X_0$ are not independent. This probability can be computed to be

$$r_1(\epsilon) = \frac{24e^{\frac{3\epsilon}{4}} - 1 + 8e^{\frac{\epsilon}{4}} + 21e^{\frac{\epsilon}{2}}}{48e^{\frac{3\epsilon}{4}}}.$$

Similarly, when the input is $[1, 1]$ and the output is $[\bot, \top]$, the probability is given by

$$r_2(\epsilon) = \frac{-22 + 32e^{\frac{\epsilon}{4}} - 3\epsilon}{48e^{\frac{\epsilon}{2}}}.$$

Observe that $r_1(\epsilon)$ and $r_2(\epsilon)$ are functions of ϵ, and hence the probabilities of outputting $[\bot, \top]$ on inputs $[0, 1]$ and $[1, 1]$ vary with ϵ. Our immediate challenge is to automatically compute expressions like $r_1(\epsilon), r_2(\epsilon)$ from the given program, the adjacent inputs, and outputs. Note that this example involves sampling from continuous distributions and is a function of ϵ. Nevertheless, we shall establish that (see Section 6 and Theorem 6.3) that for several programs, the former can be accomplished by interpreting the program as a finite-state DTMC whose transition probabilities are functions parameterized by ϵ even when the randomized choices involve infinite-support random variables. The set of programs that we identify (Section 6) is rich enough to model the most known differential privacy mechanisms when restricted to finite input and output sets.

Having computed such expressions, checking ϵ-differential privacy requires one to determine if

$$\begin{array}{ll} \text{for all} & \epsilon > 0.\ (r_1(\epsilon) \leq e^{\epsilon} r_2(\epsilon)) \\ \text{and for all} & \epsilon > 0.\ (r_2(\epsilon) \leq e^{\epsilon} r_1(\epsilon)). \end{array}$$

Note that the particular condition for the SVT example under consideration above is encodable as a first-order sentence with exponentials, and thus checking the formula for the example reduces to determining if such a first-order sentence is valid for reals, with the standard interpretation of multiplication, addition, and exponentiation. Whether there is a decision procedure that can determine the truth of first-order sentences involving real arithmetic with exponentials, is a long-standing open problem. However, a decidable fragment of such an extended first-order theory has been identified by McCallum and Weispfenning [28]. The formula for the considered example lies in this fragment. Indeed, we can show that all the formulas for the SVT example lie in this fragment. This observation presents a challenge, namely, what guarantees do we have that checking differential privacy is reducible to this decidable fragment. Indeed, we shall establish that the set of formulas that arise from the class of programs with finite-state DTMC semantics in Theorem 6.3 also lead to formulas in the same decidable fragment.

Remark. Notice that if one can compute expressions for the probability producing individual outputs on a given input, we could also check (ϵ, δ)-differential privacy, instead of just ϵ-differential privacy. The only change would be to account for δ in our constraints, and to consider all possible subsets of outputs, instead of just individual output values. Thus, the methods proposed here go beyond the scope of most automated approaches, which are restricted to vanilla ϵ-differential privacy.

4 Preliminaries

In this section, we formally define the problem of differential privacy verification that we consider in this paper and also introduce the decidable fragment of real arithmetic with exponentiation that plays a crucial role in our decision procedure. The set of reals/positive reals/rationals/positive rationals shall be denoted by $\mathbb{R}/\mathbb{R}^{>0}/\mathbb{Q}/\mathbb{Q}^{>0}$ respectively.

4.1 The Computational Problem

As illustrated by the example in Section 3, a differential privacy mechanism is typically a randomized program P_ϵ parametrized by a variable ϵ. Having a parameterized program P_ϵ captures the fact that the program's behavior depends on the privacy budget ϵ, intending to guarantee that P_ϵ is $(f(\epsilon), g(\epsilon))$-differentially private, where f and g are some functions of ϵ. The parameter ϵ is assumed to belong to some interval $I \subseteq \mathbb{R}^{>0}$ with rational end-points; usually, we take ϵ to just belong to the interval $(0, \infty)$. The program P_ϵ shall be assumed to terminate with probability 1 for every value of ϵ (in the appropriate interval).

The randomized program P_ϵ takes inputs from a set $\mathcal{U}$ and produces output in a set $\mathcal{V}$. In this paper, we shall assume that both $\mathcal{U}$ and $\mathcal{V}$ are *finite* sets that can be effectively enumerated. Despite our restriction to finite input and output sets, the computational problem of checking differential privacy is challenging (see Section 5.3). At the same time, the decidable subclass we identify (Section 6) is rich enough to model most differential privacy mechanisms when restricted to finite input and output sets. Extending our decidability results to subclasses of programs that have infinite input and output sets, is a non-trivial open problem at this time.

The computational problems we consider in this paper are as follows. Since our programs take inputs from a finite set $\mathcal{U}$, we assume that the adjacency relation $\Phi \subseteq \mathcal{U} \times \mathcal{U}$ is given as an explicit list of pairs. In general, when discussing (ϵ, δ)-differential privacy of some mechanism, the error parameter δ needs to be a function of ϵ. To define the computational problem of checking differential privacy, the function $\delta : \mathbb{R}^{>0} \to [0, 1]$ must be given as input. We, therefore, assume that this function δ has some finite representation; if $\delta(\epsilon)$ is the constant δ_0 (which is often the case), then we represent δ simply by the number δ_0. There are two computational problems we consider in this paper.

Fixed Parameter Differential Privacy Given a program P_ϵ over inputs $\mathcal{U}$ and outputs $\mathcal{V}$, adjacency relation $\Phi \subseteq \mathcal{U} \times \mathcal{U}$, and positive rational numbers $\epsilon_0, \delta_0, t \in \mathbb{Q}^{>0}$, determine if P_{ϵ_0} is $(t\epsilon_0, \delta_0)$-differentially private with respect to Φ.

Differential Privacy Given a program P_ϵ over inputs $\mathcal{U}$ and outputs $\mathcal{V}$, interval $I \subseteq \mathbb{R}^{>0}$ with rational end-points, $\delta : \mathbb{R}^{>0} \to [0, 1]$, an adjacency relation $\Phi \subseteq \mathcal{U} \times \mathcal{U}$, and a rational number $t \in \mathbb{Q}^{>0}$, determine if

P_ϵ is $(t\epsilon, \delta(\epsilon))$-differentially private with respect to Φ *for every* $\epsilon \in I$.

Observe that the Fixed Parameter Differential Privacy problem can be trivially reduced to the Differential Privacy problem by considering the singleton interval $I = [\epsilon_0, \epsilon_0]$ and $\delta(\epsilon) = \delta_0$, where the goal is to check fixed parameter differential privacy for constant privacy budget ϵ_0 and error parameter δ_0. Thus, an algorithm for checking Differential Privacy can be used to solve Fixed Parameter Differential Privacy. Unfortunately, the Fixed Parameter Differential Privacy problem is extremely challenging even when restricted to finite input and output sets— we show that it is undecidable (Section 5.3), and therefore, so is the Differential Privacy problem. We shall identify a class of programs (Section 6) for which the Differential Privacy problem (and therefore the Fixed Parameter Differential Privacy problem) is decidable.

When the differential privacy does not hold, we would like to output a counter-example.

Definition 4.1. A counter-example of (ϵ, δ) differential privacy for P_ϵ, with respect to an adjacency relation Φ, a function $\delta : \mathbb{R}^{>0} \to [0, 1]$ and a value $t \in \mathbb{Q}^{>0}$, is a quadruple $(\mathbf{in}, \mathbf{in}', O, \epsilon_0)$ such that $(\mathbf{in}, \mathbf{in}') \in \Phi$, $O \subseteq \mathcal{V}$ and $\epsilon_0 > 0$ and

$$\mathsf{Prob}(P_{\epsilon_0}(\mathbf{in}) \in O) > e^{t\epsilon_0} \mathsf{Prob}(P(\mathbf{in}') \in O) + \delta(\epsilon_0)$$

When δ is the constant function 0, then O is $\{\mathbf{out}\}$ for some $\mathbf{out} \in \mathcal{V}$.

Remark. For the rest of the paper, unless otherwise stated, we shall assume that the interval $I \subseteq \mathbb{R}^{>0}$ that contains the set of admissible ϵs is the interval $(0, \infty)$. In our paper, ϵ refers to the parameter in program P_ϵ and not the privacy budget. In our case, the privacy budget is $(t\epsilon)$. For example, some differential privacy algorithms P_ϵ are designed to satisfy $(\frac{\epsilon}{2}, 0)$-differential privacy, and so in this case t would be $\frac{1}{2}$. In the standard differential privacy definition, "ϵ" refers to the privacy budget, and so t does not appear. However, many theorems for differential privacy algorithms use "ϵ" as the program parameter, and then the privacy theorem is stated as the program being $(t\epsilon, \delta)$-differentially private. In most such cases, such a theorem is equivalent to saying that the program $P_{\frac{\epsilon}{t}}$ (obtained by replacing ϵ by $\frac{\epsilon}{t}$) is $(\epsilon, \delta(\frac{\epsilon}{t}))$-differentially private.

4.2 Reals with exponentials

As outlined in Section 3, our approach towards deciding differential privacy shall rely on reducing the question to the problem of checking the truth of a first-order sentence for the reals. Because of the definition of differential privacy, the constructed first-order sentence shall involve exponentials. It is a long-standing open problem whether there is a decision procedure for the first-order theory of reals with exponentials. However, some fragments of this theory are known to be decidable. In particular, there is a fragment identified by McCallum and Weispfenning [28], that we shall exploit in our results.

We will consider first-order formulas over a restricted signature and vocabulary. We will denote this collection of formulas as the language $\mathcal{L}_{\mathsf{exp}}$. Formulas in $\mathcal{L}_{\mathsf{exp}}$ are built using variables $\{\epsilon\} \cup \{x_i \mid i \in \mathbb{N}\}$, constant symbols 0, 1, unary function symbol $e^{(\cdot)}$ applied only to the variable ϵ, binary function symbols $+, -, \times$, and binary relation symbols $=, <$. The terms in the language are integral polynomials with rational coefficients over the variables $\{\epsilon\} \cup \{x_i \mid i \in \mathbb{N}\} \cup \{e^\epsilon\}$. Atomic formulas in the language are of the form $t = 0$ or $t < 0$ or $0 < t$, where t is a term. Quantifier free formulas are Boolean combinations of atomic formulas. Sentences in $\mathcal{L}_{\mathsf{exp}}$ are formulas of the form

$$Q\epsilon Q_1 x_1 \cdots Q_n x_n \psi(\epsilon, x_1, \ldots, x_n)$$

where ψ is a quantifier free formula, and Q, Q_is are quantifiers. In other words, sentences are formulas in prenex form, where all variables are quantified, and the outermost quantifier is for the special variable ϵ.

The theory $\mathsf{Th}_{\mathsf{exp}}$ is the collection of all sentences in $\mathcal{L}_{\mathsf{exp}}$ that are valid in the structure $\langle \mathbb{R}, 0, 1, e^{(\cdot)}, +, -, \times, =, < \rangle$, where the interpretation for $0, 1, +, -, \times$ is the standard one on reals, and e is Euler's constant; notice that this is an extension of the first-order theory of reals. The crucial property about this theory is that it is decidable.

Theorem 4.2 (McCallum-Weispfenning [28])**.** $\mathsf{Th}_{\mathsf{exp}}$ *is decidable.*

Finally, our tractable restrictions (and our proofs of decidability) shall often utilize the notion of functions *definable* in $\mathsf{Th}_{\mathsf{exp}}$; we, therefore, conclude this section with its formal definition.

Definition 4.3. A function $f : (0, \infty) \to \mathbb{R}$ is said to be *definable* in $\mathsf{Th}_{\mathsf{exp}}$, if there is a formula $\varphi_f(\epsilon, x)$ in $\mathcal{L}_{\mathsf{exp}}$ with two free variables (ϵ and x) such that

for all $a \in (0, \infty)$. $f(a) = b$ iff
$\langle \mathbb{R}, 0, 1, e^{(\cdot)}, +, -, \times, =, < \rangle \models \varphi_f(\epsilon, x)[\epsilon \mapsto a, x \mapsto b]$

5 Program syntax and semantics

We consider randomized algorithms written as simple probabilistic while programs. We introduce the syntax of these programs, along with their "natural" semantics given using Markov kernels [14, 30]. We show that the problem of checking differential privacy is undecidable for these programs.

5.1 Syntax of Simple programs

We introduce a class of programs we call Simple. Programs in Simple are probabilistic while programs in which variables can be assigned values by drawing from distributions typically used in differential privacy algorithms. Programs in Simple obey some syntactic restrictions; these syntactic

Expressions ($\mathsf{b} \in \mathcal{B}, \mathsf{x} \in \mathcal{X}, \mathsf{z} \in \mathcal{Z}, \mathsf{r} \in \mathcal{R}, d \in \mathrm{DOM}, i \in \mathbb{Z}, q \in \mathbb{Q}, g \in \mathcal{F}_{Bool}, f \in \mathcal{F}_{\mathrm{DOM}}$):

$$\begin{array}{lcl} B & ::= & \mathsf{true} \mid \mathsf{false} \mid \mathsf{b} \mid not(B) \mid B \ and \ B \mid B \ or \ B \mid g(\tilde{E}) \\ E & ::= & d \mid \mathsf{x} \mid f(\tilde{E}) \\ Z & ::= & \mathsf{z} \mid iZ \mid EZ \mid Z + Z \mid Z + i \mid Z + E \\ R & ::= & \mathsf{r} \mid qR \mid ER \mid R + R \mid R + q \mid R + E \end{array}$$

Basic Program Statements ($a \in \mathbb{Q}^{>0}$, $\sim \in \{<, >, =, \leq, \geq\}$, F is a scoring function and choose is a user-defined distribution):

$$\begin{array}{lcl} s & ::= & \mathsf{x} \leftarrow E \mid \mathsf{z} \leftarrow Z \mid \mathsf{r} \leftarrow R \mid \mathsf{b} \leftarrow B \mid \mathsf{b} \leftarrow Z_1 \sim Z_2 \mid \\ & & \mathsf{b} \leftarrow Z \sim E \mid \mathsf{b} \leftarrow R_1 \sim R_2 \mid \mathsf{b} \leftarrow R \sim E \mid \\ & & \mathsf{r} \leftarrow \mathsf{Lap}(a\epsilon, E) \mid \mathsf{z} \leftarrow \mathsf{DLap}(a\epsilon, E) \mid \\ & & \mathsf{x} \leftarrow \mathsf{Exp}(a\epsilon, F(\tilde{\mathsf{x}}), E) \mid \mathsf{x} \leftarrow \mathsf{choose}(a\epsilon, \tilde{E}) \mid \\ & & \mathsf{if}\ B\ \mathsf{then}\ P\ \mathsf{else}\ P\ \mathsf{end} \mid \mathsf{While}\ B\ \mathsf{do}\ P\ \mathsf{end} \mid \mathsf{exit} \end{array}$$

Program Statements ($\ell \in \mathsf{Labels}$)

$$P \quad ::= \quad \ell : \ s \mid \ell : \ s \, ; \, P$$

Figure 1. BNF grammar for Simple. DOM is a finite discrete domain. $\mathcal{F}_{Bool}$, ($\mathcal{F}_{\mathrm{DOM}}$ resp) are set of functions that output Boolean values (DOM respectively). $\mathcal{B}$, $\mathcal{X}$, $\mathcal{Z}$, $\mathcal{R}$ are the sets of Boolean variables, DOM variables, integer random variables and real random variables. Labels is a set of program labels. For a syntactic class S, $\tilde{S}$ denotes a sequence of elements from S. DiPWhile (see Section 6) is the subclass of programs in which the assignments to real and integer variables do not occur with the scope of a while statement.

restrictions are introduced to make it easier to describe the decidable fragment in Section 6. Despite these restrictions, the problem of checking differential privacy is undecidable for the language introduced here.

The formal syntax of Simple programs is shown in Figure 1. Programs have four types of variables: $Bool = \{\mathsf{true}, \mathsf{false}\}$; finite domain DOM [2] that we assume (without loss of generality) to be $\{-N_{\max}, \ldots 0, 1, \ldots N_{\max}\}$, a finite subset of integers [3]; reals $\mathbb{R}$; and integers $\mathbb{Z}$. The set of Boolean/DOM/integer/real program variables are respectively denoted by $\mathcal{B}/\mathcal{X}/\mathcal{Z}/\mathcal{R}$. The set of Boolean/DOM/integer/real expressions is given by the non-terminal $B/E/Z/R$ in Figure 1. We now explain the rules for such expressions. Boolean expressions (B) can be built using Boolean variables and constants, standard Boolean operations, and by applying functions from $\mathcal{F}_{Bool}$. $\mathcal{F}_{Bool}$ is assumed to be a collection of *computable* functions returning a *Bool*. We assume that $\mathcal{F}_{Bool}$ always contains a function $\mathsf{EQ}(x_1, x_2)$ that returns true iff x_1 and x_2 are equal. DOM expressions (E) are similarly built from DOM variables, values in DOM, and applying functions from set of computable functions $\mathcal{F}_{\mathrm{DOM}}$. Next, integer expressions (Z) are built using multiplication and addition with integer constants and DOM expressions, and additions with other integer expressions. Finally, real expressions (R) are built using multiplication and addition with rational constants and DOM expressions, and additions with other real-valued expressions. Notice that integer-valued expressions cannot be added or multiplied, in real-valued expressions; this syntactic restriction shall be useful later.

A program in Simple is a triple consisting of a set of (private) input variables, a set of (public) output variables, and a finite sequence of labeled statements (non-terminal P in Figure 1). The private input variables and public output variables take values from the domain DOM. Thus, the set of possibles inputs/outputs ($\mathcal{U}/\mathcal{V}$), is identified with the set of valuations for input/output variables; a valuation over a set of variables $X' = \{\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_m\} \subseteq \mathcal{X}$ is a function from X' to DOM. Note that if we represent the set X' as a sequence $\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_m$ then a valuation val over X' can be viewed as a sequence $val(\mathsf{x}_1), val(\mathsf{x}_2), \ldots, val(\mathsf{x}_m)$ of DOM elements.

We assume every statement in our program is uniquely labeled from a set of labels called Labels. Basic program statements (non-terminal s) can either be assignments, conditionals, while loops, or exit. Statements other than assignments are self-explanatory. The syntax of assignments is designed to follow a strict discipline. Real and integer variables can either be assigned the value of real/integer expression or samples drawn using the Laplace or discrete Laplace mechanism. DOM variables are either assigned values of DOM expressions or values drawn either using an exponential mechanism ($\mathsf{Exp}(a\epsilon, F(\tilde{\mathsf{x}}), E)$) or a user-defined distribution ($\mathsf{choose}(a\epsilon, \tilde{E})$). For the exponential mechanism, we require that the scoring function F be computable and return a rational value. Both of these restrictions are unlikely to be severe in practice. In the case of the user defined distribution, we demand that the probability with which a value d in DOM is chosen (as a function of the privacy budget ϵ), be definable in $\mathsf{Th}_{\mathsf{exp}}$, and that there is an algorithm that on input $a, \tilde{d}, v$ returns the formula defining the probability of sampling $d \in \mathrm{DOM}$ from the distribution $\mathsf{choose}(a\epsilon, \tilde{d})$ where $\tilde{d}$ is a sequence of values from DOM. This restriction is exploited in Section 6 to get decidability for a sub-fragment.

Finally, we consider assignments to Boolean variables. The interesting cases are those where the Boolean variable stores the result of the comparison of two expressions. The syntax does not allow for comparing real and integer expressions. This restriction is exploited later in Section 6 when the decidable fragment is identified. Finally, we will assume that in any execution, if a variable appears on the right side of an assignment statement, then it should have been assigned a value before. This assumption is not restrictive but is technically convenient when defining the semantics for programs.

[2] Though not necessary to distinguish between Booleans and finite domains, having such a distinction makes our future technical development easier.

[3] Our decidability results also hold if DOM is taken to be a finite subset of the rationals.

5.2 Markov Kernel Semantics

We briefly sketch a "natural" semantics for Simple using Markov kernels. A key step in proving our decidability result is to define a semantics using finite-state (parametrized) DTMCs for the sub-fragment DiPWhile defined in Section 6. The DTMC semantics may not seem natural on first reading. The point of the semantics in this section is, therefore, to argue the correctness of our decision procedure on the basis of the equivalence of these two semantics for DiPWhile (Sections 6 and 7). The details for this section are omitted due of space constraints and because understanding this semantics is not critical to our decidability proof. The omitted details can be found [4].

Given a fixed $\epsilon > 0$, the states in the Markov kernel-based semantics for a program P_ϵ will be of the form $(\ell, h_{Bool}, h_{\mathsf{DOM}}, h_{\mathbb{Z}}, h_{\mathbb{R}})$, where ℓ is the label of the statement of P_ϵ to be executed next, the functions h_{Bool}, h_{DOM}, $h_{\mathbb{Z}}$ and $h_{\mathbb{R}}$ assign values to the Boolean, DOM, real and integer variables of the program P_ϵ respectively. Given an input state **in**, the initial state will correspond to one where DOM-valued input variables get the values given in **in**, and all other variables either get false or 0, depending on their type. Observe that for a program P_ϵ with k program statements, i Boolean variables, j DOM variables, s integer variables, t real variables a state $(\ell, h_{Bool}, h_{\mathsf{DOM}}, h_{\mathbb{Z}}, h_{\mathbb{R}})$ can be uniquely identified with an element of the set $D_{P_\epsilon} = \{1, \ldots, k\} \times \mathcal{F}^i_{Bool} \times \mathsf{DOM}^j \times \mathbb{Z}^s \times \mathbb{R}^t$. The "natural" Borel σ-algebra on D_{P_ϵ} induces a σ-algebra on the states of P_ϵ.

The semantics of Simple programs can be defined as a Markov kernel over this σ-algebra on states. Intuitively, the Markov kernel K_ϵ corresponding to a program P_ϵ is such that for a state s and a measurable set of states C, $K_\epsilon(s, C)$ is the probability of transitioning to a state in C from s. The precise definition of this Markov kernel can be found in [4].

Executions are just sequences of states, and the σ-field on executions is the product of the σ-field on states. The Markov kernel defines a probability measure on this σ-field. Given all these observations, we take $\mathsf{Prob}_{natural}(P_\epsilon(\mathbf{in}) = \mathbf{out})$ to denote the probability (as defined by the Markov kernel of P_ϵ) of the set of all executions that start in the initial state corresponding to **in** and end in an exit state with **out** as the valuation of output variables. For the rest of the paper, we will assume that our programs terminate with probability 1.

5.3 Undecidability

The problem of checking differential privacy for Simple programs is undecidable.

Theorem 5.1. *The Fixed Parameter Differential Privacy problem and the Differential Privacy problem for programs P_ϵ in* Simple *is undecidable.*

The proof of Theorem 5.1 reduces the non-halting problem for deterministic 2-counter Minsky machines to the Fixed Parameter Differential Privacy problem. More precisely, we show that given a 2-counter Minsky machine $\mathcal{M}$ (with no input), there is a program $P^{\mathcal{M}}_\epsilon \in$ Simple such that

- $P^{\mathcal{M}}_\epsilon$ has only one input $\mathsf{x_{in}}$ and one output $\mathsf{x_{out}}$ taking values in $\mathsf{DOM} = \{0, 1\}$;
- $P^{\mathcal{M}}_\epsilon$ terminates with probability 1 for all $\epsilon \in \mathbb{R}^{>0}$;
- $P^{\mathcal{M}}_\epsilon$ is $(\epsilon, 0)$-differentially private with respect to the adjacency relation $\Phi = \{(0, 1), (1, 0)\}$ if and only if $\mathcal{M}$ does not halt.

This construction shows that Differential Privacy is undecidable. Undecidability of Fixed Parameter Differential Privacy is obtained by taking ϵ to be any constant rational number, say $\frac{1}{2}$. The formal details of the reduction are in [4].

6 DiPWhile: A decidable class of programs

We now discuss a restricted class of programs, for which we can establish decidability of checking differential privacy. The class of programs that we consider are exactly those programs in Simple that satisfy the following restriction:

Bounded Assignments We do not allow assignments to real and integer variables within the scope of a while loop. This restriction ensures that assignments to such variables happen only a *bounded* number of times during execution. Thus, without loss of generality, we assume that real and integer variables are assigned *at most once* as a program with multiple assignments to a single real and variables can always be rewritten to an equivalent program with each assignment to a variable being an assignment to a fresh variable.

We refer to this restricted class as DiPWhile. The DiPWhile language is surprisingly expressive — many known randomized algorithms for differential privacy can be encoded. We give an example of such encodings in DiPWhile. We omit labels of program statements unless they are needed.

Example 6.1. Algorithm 2 shows how SVT can be encoded in our language with $T = 0, \Delta = 1, N = 2, c = 1$. In the example we are modeling $\bot$ by 0 and $\top$ by 1. Though for-loops are not part of our program syntax, they can modeled as while loops, or if bounded (like here), they can be unrolled.

We can also encode the standard exponential distribution in DiPWhile (See [4]). Other examples that can be encoded in our language (and for which the decision procedure applies) include randomized response, the private smart sum algorithm [10] with finite discretization of output space (See Section 7.1), and private vertex cover [23].

The decidability of checking differential privacy for DiPWhile shall rely on two observations. First, the semantics of DiPWhile programs can also be defined as finite-state discrete-time Markov chains (DTMC), albeit with transition probabilities parameterized by ϵ. This observation is surprising because DiPWhile programs have real and integer

Input: q_1, q_2
Output: out_1, out_2

1 $T \leftarrow 0$;
2 $out_1 \leftarrow 0$;
3 $out_2 \leftarrow 0$;
4 $\mathsf{r}_T \leftarrow \mathsf{Lap}(\frac{\epsilon}{2}, T)$;
5 $\mathsf{r}_1 \leftarrow \mathsf{Lap}(\frac{\epsilon}{4}, q_1)$;
6 $\mathsf{b} \leftarrow \mathsf{r}_1 \geq \mathsf{r}_T$;
7 if b then
8 $\quad out_1 \leftarrow 1$
else
9 $\quad \mathsf{r}_2 \leftarrow \mathsf{Lap}(\frac{\epsilon}{4}, q_2)$;
10 $\quad \mathsf{b} \leftarrow \mathsf{r}_2 \geq \mathsf{r}_T$;
11 $\quad$ if b then
12 $\quad\quad out_2 \leftarrow 1$
$\quad$ end
end
13 exit

Algorithm 2: SVT for 1-sensitive queries with $N = 2$,$c = 1$ and $T = 0$. The numbers at the beginning of a line indicate the label of the statement.

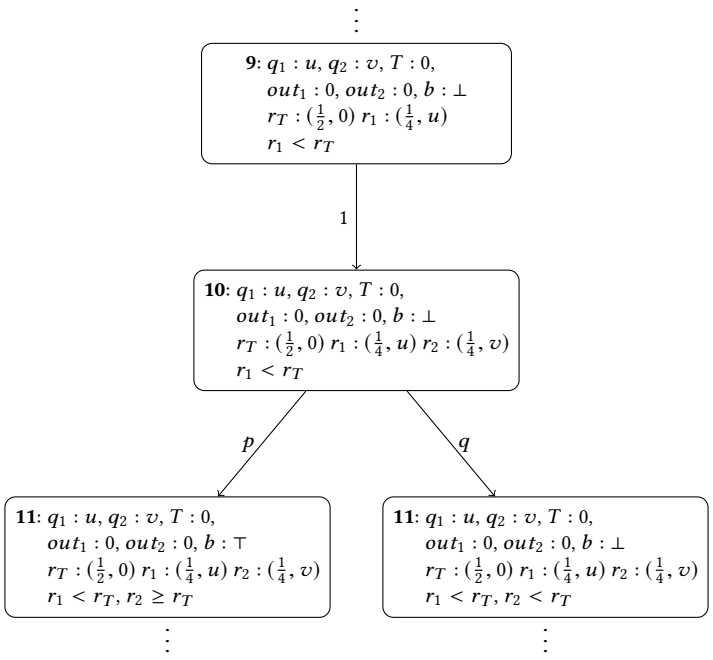

Figure 2. Partial DTMC semantics of Algorithm 2 showing the steps when lines 9 and 10 are executed. q_1 and q_2 are assumed to have values u and v, respectively. Only values of assigned program variables is shown. Third line in state shows parameters for the real values that were sampled. Last line shows the accumulated set of Boolean conditions that hold on the path.

values variables, and so the natural semantics has uncountably many states (See Section 5.2). The key insight in establishing this observation is that an equivalent semantics of DiPWhile programs can be defined without explicitly tracking the values of real and integer-valued variables. Second, all the transition probabilities arising in our semantics are definable in $\mathsf{Th}_{\mathsf{exp}}$. These two observations allow us to to establish decidability of checking differential privacy of DiPWhile programs. The rest of the section is devoted to establishing these observations. We start by formally defining *parametrized DTMCs*.

6.1 Parameterized DTMCs

Definition 6.2. A *parametrized DTMC* is a pair $\mathcal{D} = (Z, \Delta)$, where Z is a (countable) set of states, and $\Delta : Z \times Z \rightarrow (\mathbb{R}^{>0} \rightarrow [0, 1])$ is the *probabilistic transition function*. For any pair of states z, z', Δ returns a function from $\mathbb{R}^{>0}$ to $[0, 1]$, such that for every $\epsilon > 0$, $\sum_{z' \in Z} \Delta(z, z')(\epsilon) = 1$. We shall call $\Delta(z, z')$ as the probability of transitioning from z to z'.

A *definable* parametrized DTMC is a parametrized DTMC $\mathcal{D} = (Z, \Delta)$ such that for every pair of states $z, z' \in Z$, the function $\Delta(z, z')$ is definable in $\mathsf{Th}_{\mathsf{exp}}$.

A parametrized DTMC associates with each (finite) sequence of states $\rho = z_0, z_1, \ldots z_m$, a function $\mathsf{Prob}(\rho) : \mathbb{R}^{>0} \rightarrow [0, 1]$ that given an $\epsilon > 0$, returns the probability of the sequence ρ when the parameter's value is fixed to ϵ, i.e., $\mathsf{Prob}(\rho)(\epsilon) = \prod_{i=0}^{m-1} \Delta(z_i, z_{i+1})(\epsilon)$. For a state z_0 and a set of states $Z' \subseteq Z$, once again we have a function that given a value ϵ for the parameter, returns the probability of reaching Z' from z_0. This can be formally defined as $\mathsf{Prob}(z_0, Z')(\epsilon) = \sum_{\rho \in z_0(Z \setminus Z')^* Z'} \mathsf{Prob}(\rho)(\epsilon)$. In other words, $\mathsf{Prob}(z_0, Z')(\epsilon)$ is the sum of the probability of all sequences starting in z_0, ending in Z', such that no state except the last is in Z'.

6.2 Parametrized DTMC semantics of DiPWhile

The parametrized DTMC semantics of a DiPWhile program P_ϵ shall be denoted as $[\![P_\epsilon]\!]$. We describe $[\![P_\epsilon]\!]$ informally here. As mentioned above, the key insight in defining the semantics of a DiPWhile program as a finite-state, parametrized DTMC, is that the actual values of real and integer variables need not be tracked. A state of $[\![P_\epsilon]\!]$ is going to be a tuple of the form $(\ell, f_{Bool}, f_{\mathsf{DOM}}, f_{\mathsf{int}}, f_{\mathsf{real}}, C)$ where ℓ is the label of the statement of P_ϵ to be executed next. $[\![P_\epsilon]\!]$ is an abstraction of the set of all concrete states that are compatible with it. The partial functions f_{Bool} and f_{DOM} assign values to the *Bool* and DOM variables, respectively; this is just like in the natural semantics.

Let us now look at the partial function f_{real}. Intuitively, f_{real} is supposed to be the "valuation" for the real variables. But instead of mapping each variable to a *concrete* value in $\mathbb{R}$, we shall instead map it into a finite set. To understand this mapping, let us recall that in DiPWhile, a real variable is assigned only once in a program. Further, such an assignment

either assigns the value of a linear expression over program variables, or a value sampled using a Laplace mechanism. In the former case, f_{real} maps a variable to the linear expression it is assigned; and in the latter case, the value of the parameters of the Laplace mechanism used in sampling. In the latter case, since the first parameter is always of the form $a\epsilon$, we need to note only a in the mapping. Notice that the range of f_{real} is now a finite set as P_ϵ contains only a finite number of linear expressions, and the parameters of sampled Laplacian take values from the finite set DOM. Similarly, the partial function f_{int} maps each integer variable to either the linear expression it is assigned or the parameters of the sampled discrete Laplace mechanism. The last state component C is the set of Boolean conditions on real and integer variables that hold along the path thus far; this shall become clearer when we describe the transitions. Since the Boolean conditions must be Boolean expressions in the program or their negation, C is also a finite set. These observations show that $[\![P_\epsilon]\!]$ has finitely many states. Intuitively, a state of $[\![P_\epsilon]\!]$ is an abstraction of the set of all concrete states that respect the Boolean conditions in C and the constraints imposed by assignments of real and integer expressions to real and integer variables, respectively.

We now sketch how the state is updated in $[\![P_\epsilon]\!]$. Updates to DOM variables shall be as expected — it shall be a probabilistic transition if the assignment samples using an exponential mechanism or a user-defined distribution, and it shall be a deterministic step updating f_{DOM} otherwise. Assignments to real variables are *always deterministic* steps that change the function f_{real}. Thus, even if the step samples using the Laplace mechanism, in the semantics, it shall be modeled as a deterministic step where f_{real} is updated by storing the parameters of the distribution. Similarly, all integer assignments are deterministic steps as well.

The assignment of a Boolean expression to a Boolean variable is as expected — we update the valuation f_{Bool} to reflect the assignment. The unexpected case is $\mathsf{b} \leftarrow R_1 \sim R_2$ when a boolean variable gets assigned the result of the comparison of two real expressions; the case of comparing two integer expressions is similar. In this case, if the probability of C holding is 0, then our construction will ensure that this state is not reachable with non-zero probability. Otherwise, we transition to a state where $R_1 \sim R_2$ is added to C with probability equal to the probability that $(R_1 \sim R_2)$ holds conditioned on the fact that C holds, and with the remaining probability, we shall transition to the state where $\neg(R_1 \sim R_2)$ is added to C. Thus, Boolean assignments which compare integer and real variables are modeled by probabilistic transitions. Finally, branches and while loop conditions are deterministic steps, with the value of the Boolean variable (of the condition) in f_{Bool} determining the choice of the next statement.

Let $\mathsf{Prob}_{DTMC}(P_\epsilon(\mathbf{in}) = \mathbf{out})$ denote the probability that P_ϵ outputs value **out** on the input **in** under the DTMC semantics. This is just the probability of reaching an exit state with **out** as valuation of output variables from the initial state with **in** as the valuation of input variables. We can show that this probability is the same as the probability $\mathsf{Prob}_{natural}(P_\epsilon(\mathbf{in}) = \mathbf{out})$ obtained by the natural semantics discussed above.

It is worth noting how key syntactic restrictions in DiPWhile programs play a role in defining its semantics. The first restriction is that integer and real variables are not assigned in the scope of a while loop. This restriction is critical to ensure that the DTMC $[\![P_\epsilon]\!]$ is finite-state. Since we track distribution parameters and linear expressions for such variables, this restriction ensures that we only remember a bounded number of these. Second, DiPWhile disallows a comparison between real and integer expressions in its syntax. Recall that such comparison steps result in a probabilistic transition, where we compute the probability of the comparison holding conditioned on the properties in C holding. It is unclear if a closed-form expression for such probabilities can be computed when integer and real random variables are compared. Hence such comparisons are disallowed.

Probabilistic transitions in our semantics arise due to two reasons. First are assignments to DOM variables that sample according to either the exponential or a user-defined distribution. The resulting probabilities are easily seen to be definable in $\mathsf{Th}_{\mathsf{exp}}$. The second is due to comparisons between real and integer expressions. We can prove that in this case also, the resulting probabilities are definable in $\mathsf{Th}_{\mathsf{exp}}$; this proof is non-trivial, and can be found in [4]. All these observations together give us the following theorem.

Theorem 6.3. *For any* DiPWhile *program* P_ϵ, $[\![P_\epsilon]\!]$ *is a finite, definable, parametrized DTMC that is computable.*

Example 6.4. The parametrized DTMC semantics of Algorithm 2 is partially shown in Figure 2. We show only the transitions corresponding to executing lines 9 and 10 of the algorithm, when $q_1 = u$ and $q_2 = v$ initially; here $u, v \in \{\bot, \top\}$. The multiple lines in a given state give the different components of the state. The first two lines give the assignment to *Bool* and DOM variables, the third line gives values to the integer/real variables, and the last line is the Boolean conditions that hold along a path. Since 9 and 10 are in the else-branch, the condition $r_1 < r_T$ holds. Notice that values to real variables are not explicit values, but rather the parameters used when they were sampled. Finally, observe that probabilistic branching takes place when line 10 is executed, where the value of b is taken to be the result of comparing r_2 and r_T. The numbers p and q correspond to the probability that the conditions in a branch hold, given the parameters used to sample the real variables and *conditioned* on the event that $r_1 < r_T$.

7 Checking differential privacy for DiPWhile programs

We shall now establish that the problem of checking differential privacy for DiPWhile programs is decidable. The proof relies on the characterization of the semantics of a DiPWhile program as a finite, definable, parameterized DTMC (See Theorem 6.3). An important observation about a finite, definable, parametrized DTMC is that the probability of reaching a given set of states Z' from a given state z_0 is both definable and computable.

Lemma 7.1. *For any finite-state, definable, parametrized DTMC $\mathcal{D} = (Z, \Delta)$, any state $z_0 \in Z$ and set of states $Z' \subseteq Z$, the function* $\mathsf{Prob}(z_0, Z')$ *is definable in* $\mathsf{Th}_{\exp}$. *Moreover, there is an algorithm that computes the formula defining* $\mathsf{Prob}(z_0, Z')$.

The proof of Lemma 7.1 exploits the connection between reachability probabilities in DTMCs and linear programming [2, 32]; details are in [4]. The main result of the paper now follows from Theorem 6.3 and Lemma 7.1.

Theorem 7.2. *The Fixed Parameter Differential Privacy and Differential Privacy problems are decidable for* DiPWhile *programs P_ϵ, rational numbers $t \in \mathbb{Q}^{>0}$ and definable functions $\delta(\epsilon)$. Furthermore, if P_ϵ is not $(t\epsilon, \delta)$ differentially private for some rational number t and admissible value of ϵ then we can compute a counter-example.*

Proof. Let **in** and **out** be arbitrary valuations to input and output variables, respectively. Observe that the function $\epsilon \mapsto \mathsf{Prob}(P_\epsilon(\mathbf{in}) = \mathbf{out})$ is nothing but $\mathsf{Prob}(z_0, Z')$ in $[\![P_\epsilon]\!]$, where z_0 is the initial state corresponding to valuation **in**, and Z' is the set of all terminating states that have valuation **out** for output variables. Since $[\![P_\epsilon]\!]$ (Theorem 6.3) and $\mathsf{Prob}(z_0, Z')$ (Lemma 7.1) are computable, we can construct a formula $\varphi_{\mathbf{in},\mathbf{out}}(\epsilon, x_{\mathbf{in},\mathbf{out}})$ of $\mathcal{L}_{\exp}$ that defines the function $\epsilon \mapsto \mathsf{Prob}(P_\epsilon(\mathbf{in}) = \mathbf{out})$.

Let $\varphi_\delta(\epsilon, x_\delta)$ be the formula defining the function δ. Let $t = \frac{p}{q}$ where p, q are natural numbers. Consider the sentence

$$\begin{aligned}\psi = \forall\epsilon.\forall z.[\forall x_{\mathbf{in},\mathbf{out}}]_{\mathbf{in}\in\mathcal{U},\mathbf{out}\in\mathcal{V}}.\forall x_\delta.\\ ((\epsilon > 0) \wedge (e^{p\epsilon} = z^q) \wedge (z > 0) \wedge \varphi_\delta(\epsilon, x_\delta)\\ \textstyle\bigwedge_{\mathbf{in}\in\mathcal{U},\mathbf{out}\in\mathcal{V}}\ \varphi_{\mathbf{in},\mathbf{out}}(\epsilon, x_{\mathbf{in},\mathbf{out}}))\\ \rightarrow (\textstyle\bigwedge_{(\mathbf{in}_1,\mathbf{in}_2)\in\Phi, O\subseteq\mathcal{V}}\\ \textstyle\sum_{\mathbf{out}\in O} x_{\mathbf{in}_1,\mathbf{out}} < z \sum_{\mathbf{out}\in O} x_{\mathbf{in}_2,\mathbf{out}}\ + x_\delta))\end{aligned}$$

It is easy to see P_ϵ is $(t\epsilon, \delta(\epsilon))$ differentially private for all ϵ iff ψ is true over the reals. In the syntax of $\mathcal{L}_{\exp}$, we cannot take qth roots of e; therefore, we introduce the variable z, which enables us to write the constraints using only $e^{a\epsilon}$, where $a \in \mathbb{N}$. Notice that ψ belongs to $\mathcal{L}_{\exp}$ if we convert it to prenex form. Decidability, therefore, follows from the decidability of $\mathsf{Th}_{\exp}$.

If P_ϵ is not differentially private, then the sentence ψ does not hold. The decision procedure for $\mathsf{Th}_{\exp}$ will, in this case, return an ϵ_0 that witnesses the privacy violation of P_ϵ. Using ϵ_0, the counter-example $(\mathbf{in}, \mathbf{in}', O, \epsilon_0)$ can be easily constructed by enumerating **in**, **in**′ and O. □

An easy consequence of Theorem 7.2 is that differential privacy is decidable for the subclass of program in Simple that do not have integer and real-valued variables. Let Finite DiPWhile denote this set of programs. Observe that due to the presence of While loops, Finite DiPWhile programs may still have unbounded length executions (including infinite executions).

Corollary 7.3. *The Fixed Parameter Differential Privacy and Differential Privacy problems are decidable for* Finite DiPWhile *programs P_ϵ, rational numbers $t \in \mathbb{Q}^{>0}$ and definable functions $\delta(\epsilon)$.*

We observe that our methods can be employed to analyze larger classes of programs (than just those in DiPWhile). For example, a sufficient condition to ensure the decidability is to consider programs with the property that, for each input, the probability distribution on the outputs is definable in $\mathsf{Th}_{\exp}$. We conclude the section by showing how our procedure is useful when reasoning about integer and real-valued outputs.

Remark. We sketch here how the proofs of Theorem 7.2 changes when the set of admissible ϵ is taken to be an interval I with rational end-points. Let P_ϵ, t and $\delta(\epsilon)$ be as in the proof of Theorem 7.2. When ϵ is restricted to an interval I, we will require the user-definable distributions to be definable in $\mathsf{Th}_{\exp}$ only on the interval I. As in the proof of Theorem 7.2, we can construct a formula $\varphi_{\mathbf{in},\mathbf{out}}(\epsilon, x_{\mathbf{in},\mathbf{out}})$ of $\mathcal{L}_{\exp}$ that defines the function $\epsilon \mapsto \mathsf{Prob}(P_\epsilon(\mathbf{in}) = \mathbf{out})$. For simplicity, consider the case when I be the interval $[r, s]$. Consider the sentence ψ_I that is obtained from ψ in the proof of Theorem 7.2 by replacing the subformula $(\epsilon > 0)$ by $(a \le \epsilon) \wedge (\epsilon \le b)$. Then P_ϵ is $(t\epsilon, \delta(\epsilon))$ will be differentially private for all $\epsilon \in I$ iff ψ_I is true over the reals.

7.1 Finite discretization of infinite output spaces

Our decision procedure assumes that the output space is finite. In several examples, the program outputs are reals or unbounded integers (and combinations thereof). Nevertheless, we argue that our decision procedure is useful for the verification of differential privacy in this case also. In particular, our method provides an under-approximation technique for checking the differential privacy of programs with infinite outputs. Our approach in such cases is to discretize the output space into finitely many intervals.

We illustrate this for the special case when a program P outputs the value of one real random variable, say r. Now, suppose that we modify P to output a finite discretized version of r as follows. Let $\mathsf{seq} = a_0 < a_1 < \ldots a_n$ be a sequence of rationals and let $\mathsf{Disc}_{\mathsf{seq}}(x)$ be equal to a_0 if $x \le a_0$, equal to a_i $(0 < i < n)$ if $a_{i-1} < x \le a_i$, and equal to a_n if $x > a_{n-1}$.

Consider the program $P_{\mathsf{Disc,seq}}$ that instead of outputting r, outputs $\mathsf{Disc}_{\mathsf{seq}}(\mathsf{r})$. It is easy to see that if P is differentially private then so must be $P_{\mathsf{Disc,seq}}$. Therefore, if $P_{\mathsf{Disc,seq}}$ is not differentially private then we can conclude that P is not differentially private. Thus, if our procedure finds a counter-example for $P_{\mathsf{Disc,seq}}$, then it also has proved that the program P is not differentially private. Our method is, therefore, an under-approximation technique for checking the differential privacy of P. In fact, it is a *complete* under-approximation method in the sense that P is differentially private iff for each possible seq, $P_{\mathsf{Disc,seq}}$ is differentially private.

8 Experimental evaluation

We implemented a simplified version of the algorithm, presented earlier, for proving/disproving differential privacy of DiPWhile programs. Our tool DiPC [3] handles loop-free programs, i.e., acyclic programs. Programs with bounded loops (with constant bounds) can be handled by unrolling loops. The tool takes in an input program P_ϵ parametrized by ϵ and an adjacency relation, and either proves P_ϵ to be differentially private for all ϵ or returns a counter-example. The tool can also be used to check differential privacy for a given, fixed ϵ, or to check for $k\epsilon$-differential privacy for some constant k. DiPC is implemented in C++ and uses Wolfram Mathematica®. It works in two phases — in the first phase, a Mathematica®script is produced with commands for all the output probability computations and the subsequent inequality checks and in the second phase, the generated script is run on Mathematica. Details about the tool and its design can be found in [4].

We used various examples to measure the effectiveness of our tool. These include SVT [20, 27], Noisy Maximum [17], Noisy Histogram [17] and Randomized Response [19] and their variants. Detailed descriptions of these algorithms and their variants can be found in [4].

We ran all the experiments on an octa-core Intel®Core i7-8550U @ 1.8gHz CPU with 8GB memory. The running times reported are the average of 3 runs of the tool. In the tables, T1 refers to the time needed by the C++ phase to generate the Mathematica scripts, and T2 refers to the time used by Mathematica to check the scripts. Due to space constraints, we report only a small fraction of our experiments; full details of all our experiments can be found in [4].

Salient observations about our experiments are follows.

1. DiPC successfully proves algorithms to be differentially private and finds counter-examples to demonstrate a violation of privacy in reasonable time. Table 1 shows the running time of DiPC on some examples for 3 queries. We chose to use 3 queries because for algorithms that are not private, counter-examples can be found with 3 queries.
2. The time to generate Mathematica scripts is significantly smaller than the time taken by Mathematica to check the scripts (i.e., T1 $\ll$ T2). Further, most of the time spent by Mathematica is for computing output probabilities; the time to perform comparison checks for adjacent inputs was relatively small. Thus, programs that do not use real variables (Rand2 in Table 1, for example) can be analyzed more quickly.
3. For algorithms that are not differentially private, DiPC can automatically identify the pair of inputs, output, and ϵ for which privacy is violated. Table 2, shows the results for the smallest counter-example found by DiPC for some examples. Further, counter-examples found by DiPC are much smaller, in terms of queries, than those found in [17]; the number of queries needed in the counter-examples in [17] for NMax3, NMax4, and SVT5 were 5, 5, and 10, respectively, as opposed to 3, 1, and 2 found by DiPC.
4. DiPC is the first automated tool that can check (ϵ, δ)-differential privacy. To evaluate this feature, we tested DiPC on a version of SVT, Sparse [20], which is manually proven to be $(\frac{\epsilon}{2}, \delta_{\mathsf{svt}})$-differentially private for any number of queries in [20] by using advanced composition theorems. Here δ_{svt} is a second parameter in the algorithm. In our experiments, we tested $(\frac{\epsilon}{2}, \delta_{\mathsf{svt}})$-differential privacy of Sparse with fixed values of δ_{svt} for $c = 1, 2$ and 3 queries, validating the result in [20]. As we were dealing with only 3 queries, we also managed to obtain better bounds on the error parameter.

Algorithm	Runtime (T1/T2)	ϵ-Diff. Private
SVT	0s/825s	✓
SVT2	0s/768s	✓
SVT5	0s/2s	✗
NMax4	1s/58s	✗
Rand2	0s/0s	✗

Table 1. Runtime for 3 queries for each algorithm searching over adjacency pairs and all ϵ>0, with parameters being [c=1, Δ=1, DOM={-1,0,1}, seq = $(-1 < 0 < 1)$]. For SVT, we also have T=0.

Algo	|Q|	Output	Input 1	Input 2	ϵ	Runtime (T1/T2)
SVT5	2	$[\bot\ \top]$	[-1 0]	[-1 -1]	27	0s/2s
NMax3	3	-1, seq=(-1<0<1)	[-1 -1 -1]	[0 0 0]	27	0s/310s
NMax4	1	0, seq=(-1<0<1)	[-1]	[0]	27	0s/2s
Rand2	1	$[\bot]$	$[\bot]$	$[\top]$	9/34	0s/0s

Table 2. Smallest Counter-example found for each non-differentially private algorithm, searching over all adj. pairs and $\epsilon > 0$, with parameters being [c=1, Δ=1, DOM={-1,0,1}]

9 Related work

The main thread of related work has focused on formal systems for proving that an algorithm is differentially private.

Such systems are helpful because they rule out the possibility of mistakes in privacy analyses. Starting from Reed and Pierce [31], several authors [16, 21] have proposed linear (dependent) type systems for proving differential privacy. However, it is not possible to verify some of the most advanced examples, such as a sparse vector or vertex cover, using these type systems. Moreover, type-checking and type-inference for linear (dependent) types are challenging. For example, the type checking problem for DFuzz, a language for differential privacy, is undecidable [15]. Barthe et al [5, 6, 8] develop several program logics based on probabilistic couplings for reasoning about differential privacy. These logics have been used successfully to analyze many classic examples from the literature, including the sparse vector technique. However, these logics are limited: they cannot disprove privacy; extensions may be required for specific examples; building proofs is challenging. The last issue has been addressed by a series of works that provide automated methods for proving differential privacy automatically. Zhang and Kifer [34] introduce randomness alignments as an alternative to couplings and build a dependent type system that tracks randomness alignments. Automation is then achieved by type inference. Albarghouthi and Hsu [1] propose coupling strategies, which rely on a fine-grained notion of variable approximate coupling, which draws inspiration both from approximate couplings and randomness alignment. They synthesize coupling strategies by considering an extension of Horn clauses with probabilistic coupling constraints and developing algorithms to solve such constraints. Recently Wang et al [33] develop an improved method based on the idea of shadow executions. Their approach is able to verify Sparse Vector and many other challenging examples efficiently. However, these methods are limited to vanilla ϵ-differential privacy and do not accommodate bounds that are obtained by advanced composition (since $\delta \neq 0$).

In an independent line of work, Chatzikokolakis, Gebler and Palamidessi [11] consider the problem of differential privacy for Markov chains. Later, Liu, Wang, and Zhang [26] develop a probabilistic model checking approach for verifying differential privacy properties. Their approach is based on modeling differential private programs as Markov chains. Their encoding is more direct than ours (i.e. it assumes that a finite-state Markov chain is given), and they do not provide a decision procedure with real and integer variables. Furthermore, the DTMCs are not parameterized by ϵ. Chistikov and Murawski and Purser [12, 13] propose an elegant method based on skewed Kantorovich distance for checking approximate differential privacy of Markov chains.

The dual problem is to find violations of differential privacy automatically. This is useful to help privacy practitioners discover potential problems early in the development cycle. Two recent and concurrent works by Ding et al [17] and Bischel et al [9] develop automated methods for finding privacy violations. Ding et al. propose an approach that combines purely statistical methods based on hypothesis testing and symbolic execution. Bischel et al. develop an approach based on a combination of optimization methods and language-specific techniques for computing differentiable approximations of privacy estimations. Both methods are fully automated. However, both methods can only be used for concrete numerical values of the privacy budget ϵ.

Gaboardi et. al [22] study the complexity of deciding differential privacy for randomized Boolean circuits. Their results are proved by reduction to majority problems and are incomparable with ours: the only probabilistic choices in [22] are fair coin tosses and e^{ϵ} is taken to be a fixed rational number.

10 Conclusions

We showed that the problem checking differential privacy is in general undecidable, identified an expressive sub-class of programs (DiPWhile) for which the problem is decidable, and presented the results of analyzing many known differential privacy algorithms using our tool DiPC which implements a decision procedure for DiPWhile programs. Advantages of DiPC include the ability to automatically, both prove algorithms to be private for all $\epsilon > 0$, and find counter-examples to demonstrate privacy violations. In addition DiPC can check bounds that are based on concentration inequalities, in particular bounds that use advanced composition theorems. Such bounds are out of reach of most other tools that prove privacy or search for counter-examples.

In the future, it would be interesting to extend this work to handle programs with input/output variables that take values in infinite domains, and parametrized privacy algorithms that work for an unbounded number of input and output variables. Another important problem is developing decision procedures that can prove tight accuracy bounds, and detect violations of accuracy bounds. We also plan to investigate extending the decision procedure to cover algorithms that are currently out of the scope of our decision procedure such as the multiplicative weights and iterative database construction [24, 25], and those involving Gaussian distributions.

Acknowledgments

We thank the anonymous reviewers for their useful comments. Their inputs have improved the paper, especially the presentation of the semantics. Rohit Chadha was partially supported by NSF CNS 1553548 and NSF CCF 1900924. A. Prasad Sistla was partially supported by NSF CCF 1901069 and NSF CCF 1564296. Mahesh Viswanathan was partially supported by NSF CCF 1901069.

References

[1] Aws Albarghouthi and Justin Hsu. 2018. Synthesizing coupling proofs of differential privacy. *PACMPL* 2, POPL (2018), 58:1–58:30.

[2] C. Baier and J.-P. Katoen. 2008. *Principles of Model Checking*. MIT Press.

[3] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2019. Differential Privacy Checker (DiPC). https://anonymous.4open.science/repository/febcbe47-1c53-41db-be91-ea98b4cf18c1/.

[4] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2020. Deciding Differential Privacy for Programs with Finite Inputs and Outputs. arXiv:1910.04137 [cs.CR]

[5] Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Advanced Probabilistic Couplings for Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 55–67.

[6] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Proving differential privacy via probabilistic couplings. In *IEEE Symposium on Logic in Computer Science (LICS), New York, New York*.

[7] Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin C. Pierce. 2016. Programming language techniques for differential privacy. *SIGLOG News* 3, 1 (2016), 34–53.

[8] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella-Béguelin. 2013. Probabilistic Relational Reasoning for Differential Privacy. *ACM Transactions on Programming Languages and Systems* 35, 3 (2013), 9.

[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 *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 508–524.

[10] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and continual release of statistics. *ACM Transactions on Information and System Security* 14, 3 (2011), 26.

[11] Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. 2014. Generalized Bisimulation Metrics. In *35th International Conference on Concurrency Theory, CONCUR 2014*. Springer Berlin Heidelberg, 32–46.

[12] Dmitry Chistikov, Andrzej S. Murawski, and David Purser. 2018. Bisimilarity Distances for Approximate Differential Privacy. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018 (Lecture Notes in Computer Science)*, Shuvendu K. Lahiri and Chao Wang (Eds.), Vol. 11138. Springer, 194–210.

[13] Dmitry Chistikov, Andrzej S. Murawski, and David Purser. 2019. Asymmetric Distances for Approximate Differential Privacy. In *30th International Conference on Concurrency Theory, CONCUR 2019 (LIPIcs)*, Wan Fokkink and Rob van Glabbeek (Eds.), Vol. 140. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:17.

[14] Erhan Cinlar. 2011. *Probability and Stochastics*. Springer.

[15] Arthur Azevedo de Amorim, Emilio Jesús Gallego Arias, Marco Gaboardi, and Justin Hsu. 2015. Really Natural Linear Indexed Type Checking. http://arxiv.org/abs/1503.04522. *CoRR* abs/1503.04522 (2015). arXiv:1503.04522

[16] Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2019. Probabilistic Relational Reasoning via Metrics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*. 1–19.

[17] Zeyu Ding, Yuxin Wang, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. 2018. Detecting Violations of Differential Privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 475–489.

[18] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *IACR Theory of Cryptography Conference (TCC), New York, New York*. 265–284.

[19] 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 *ACM SIGACT Symposium on Theory of Computing (STOC), Bethesda, Maryland*. 381–390.

[20] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[21] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C Pierce. 2013. Linear dependent types for differential privacy. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Rome, Italy*. 357–370.

[22] Marco Gaboardi, Kobbi Nissim, and David Purser. 2019. The Complexity of Verifying Circuits as Differentially Private. http://arxiv.org/abs/1911.03272. *CoRR* abs/1911.03272 (2019). arXiv:1911.03272 To Appear in 47th International Colloquium on Automata, Languages and Programming (ICALP' 20), 2020.

[23] Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. 2010. Differentially private combinatorial optimization. In *ACM–SIAM Symposium on Discrete Algorithms (SODA), Austin, Texas*. 1106–1125.

[24] Anupam Gupta, Aaron Roth, and Jonathan Ullman. 2012. Iterative Constructions and Private Data Release. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings (Lecture Notes in Computer Science)*, Ronald Cramer (Ed.), Vol. 7194. Springer, 339–356.

[25] Moritz Hardt and Guy N. Rothblum. 2010. A Multiplicative Weights Mechanism for Privacy-Preserving Data Analysis. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 61–70.

[26] Depeng Liu, Bow-Yaw Wang, and Lijun Zhang. 2018. Model Checking Differentially Private Properties. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018 (Lecture Notes in Computer Science)*, Sukyoung Ryu (Ed.), Vol. 11275. Springer, 394–414.

[27] Min Lyu, Dong Su, and Ninghui Li. 2017. Understanding the Sparse Vector Technique for Differential Privacy. *Proceedings of VLDB* 10, 6 (2017), 637–648.

[28] Scott McCallum and Volker Weispfenning. 2012. Deciding polynomial-transcendental problems. *Journal of Symbolic Computation* 47, 1 (2012), 16–31.

[29] Frank McSherry and Kunal Talwar. 2007. Mechanism Design via Differential Privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*. IEEE Computer Society, 94–103.

[30] Prakash Panangaden. 1999. The Category of Markov Kernels. *Electronic Notes in Theoretical Computer Science* 22 (12 1999), 171–187.

[31] Jason Reed and Benjamin C Pierce. 2010. Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy. In *ACM SIGPLAN International Conference on Functional Programming (ICFP), Baltimore, Maryland*.

[32] J. M. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. 2004. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*. AMS.

[33] Yuxin Wang, Zeyu Ding, Guanhong Wang, Daniel Kifer, and Danfeng Zhang. 2019. Proving differential privacy with shadow execution. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, (PLD)I*. 655–669.

[34] Danfeng Zhang and Daniel Kifer. 2017. LightDP: towards automating differential privacy proofs. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 888–901.