Deciding Accuracy of Differential Privacy Schemes

GILLES BARTHE, Max Planck Institute for Security and Privacy, Bochum, Germany
ROHIT CHADHA, University of Missouri, USA
PAUL KROGMEIER, University of Illinois, Urbana-Champaign, USA
A. PRASAD SISTLA, University of Illinois, Chicago, USA
MAHESH VISWANATHAN, University of Illinois, Urbana Champaign, USA

Differential privacy is a mathematical framework for developing statistical computations with provable guarantees of privacy and accuracy. In contrast to the privacy component of differential privacy, which has a clear mathematical and intuitive meaning, the accuracy component of differential privacy does not have a general accepted definition; accuracy claims of differential privacy algorithms vary from algorithm to algorithm and are not instantiations of a general definition. We identify program discontinuity as a common theme in existing *ad hoc* definitions and introduce an alternative notion of accuracy parametrized by, what we call, distance to disagreement — the distance to disagreement of an input x w.r.t. a deterministic computation f and a distance d, is the minimal distance $d(x, y)$ over all y such that $f(y) \neq f(x)$. We show that our notion of accuracy subsumes the definition used in theoretical computer science, and captures known accuracy claims for differential privacy algorithms. In fact, our general notion of accuracy helps us prove better claims in some cases. Next, we study the decidability of accuracy. We first show that accuracy is in general undecidable. Then, we define a non-trivial class of probabilistic computations for which accuracy is decidable (unconditionally, or assuming Schanuel's conjecture). We implement our decision procedure and experimentally evaluate the effectiveness of our approach for generating proofs or counterexamples of accuracy for common algorithms from the literature.

CCS Concepts: • **Security and privacy → Logic and verification**; • **Software and its engineering → Formal software verification**.

Additional Key Words and Phrases: accuracy, differential privacy, decidability

ACM Reference Format:
Gilles Barthe, Rohit Chadha, Paul Krogmeier, A. Prasad Sistla, and Mahesh Viswanathan. 2021. Deciding Accuracy of Differential Privacy Schemes. *Proc. ACM Program. Lang.* 1, POPL, Article 1 (January 2021), 35 pages.

1 INTRODUCTION

Differential privacy [Dwork et al. 2006; Dwork and Roth 2014] is a mathematical framework for performing privacy-preserving computations over sensitive data. One important feature of differential privacy algorithms is their ability to achieve provable individual privacy guarantees and at the same time ensure that the outputs are reasonably accurate. In the case of privacy, these guarantees relate executions of the differentially private algorithm on adjacent databases. These privacy guarantees are an instance of relational properties, and they have been extensively studied in the context of program verification [Albarghouthi and Hsu 2018; Barthe et al. 2020a, 2013; Gaboardi et al. 2013; Reed and Pierce 2010; Zhang and Kifer 2017] and program testing [Bichsel et al. 2018; Ding et al. 2018]. In the case of accuracy, these guarantees relate the execution of the differentially private algorithm to that of an "ideal" algorithm, which can be assumed to be deterministic. Typically, "ideal" algorithms compute the true value of a statistical computation,

Authors' addresses: Gilles Barthe, Max Planck Institute for Security and Privacy, Bochum, Germany, gjbarthe@gmail.com; Rohit Chadha, University of Missouri, USA, chadhar@missouri.edu; Paul Krogmeier, University of Illinois, Urbana-Champaign, USA, paulmk2@illinois.edu; A. Prasad Sistla, University of Illinois, Chicago, USA, sistla@uic.edu; Mahesh Viswanathan, University of Illinois, Urbana Champaign, USA, vmahesh@illinois.edu.

2021. 2475-1421/2021/1-ART1 $15.00
https://doi.org/

while differentially private algorithms compute noisy versions of the right answer. The precise relationship between the output of the "ideal" algorithm and the differentially private one varies from algorithm to algorithm, and no general definition of accuracy has been proposed in this context. In some cases, the definition of accuracy is similar to the one used in the context of randomized algorithm design [Motwani and Raghavan 1995], where we require that the output of the differentially private algorithm be close to the true output (say within distance γ) with high probability (say at least $1 - \beta$). Such a notion of accuracy is similar to computing error bounds, and there has been work on formally verifying such a definition of accuracy on some examples [Barthe et al. 2016b; Smith et al. 2019; Vesga et al. 2019]. Unfortunately, prior work on formal verification of accuracy suffers from two shortcomings:

- the lack of a general definition for accuracy: as pointed out, each differential privacy algorithm in the literature has its own specific accuracy claim that is not an instantiation of a general definition. The lack of a general definition for accuracy means that the computational problem of "verifying accuracy" has not been defined, which prevents its systematic study.
- imprecise bounds: accuracy bounds in theoretical papers are often established using concentration bounds, e.g. Chernoff bound, by hand. Existing verification frameworks, with the exception of Vesga et al. [2019], cannot be used to verify accuracy claims that are established using concentration bounds. This is due to the fact that applying concentration bounds generally requires proving independence, which is challenging. Thus, the bounds usually verified automatically (with the exception of Vesga et al. [2019]) by the existing techniques are weaker than those known in literature.

This paper overcomes both shortcomings by proposing a general notion of accuracy and by proving decidability of accuracy for a large class of algorithms that includes many differentially private algorithms from the literature.

Technical contributions. Our focus is the verification of accuracy claims for differential privacy algorithms that aim to bound the error of getting the *correct* answer. Other notions of utility or accuracy, such as those that depend on variance and other moments are out of the scope of this paper. Our contributions in this space are three-fold: (a) we give a definition of accuracy that captures accuracy claims known in the literature, (b) study the computational problem of checking accuracy, as identified by our definition of accuracy, and (c) perform an experimental evaluation of our decision procedure.

General definition of accuracy. The starting point of our work is the (well-known) observation that the usual definition used in theoretical computer science to measure the utility of a randomized algorithm (informally discussed above) fails to adequately capture the accuracy claims made for many differential privacy algorithms. To see why this is the case, consider Sparse (also called Sparse Vector Mechanism or SVT). The problem solved by Sparse is the following: given a threshold T, a database x, and a list of queries of length m, output the list of *indices* of the first c queries whose output on x is greater than or equal to T. Sparse solves this problem while maintaining the privacy of the database x by introducing noise to the query answers as well as to T when comparing them. Because of this, Sparse's answers cannot be accurate with high probability, if the answers to the query are very close to T — the correct answer is "discontinuous" near T while the probability distributions of the introduced noise are continuous functions. Thus, we can expect Sparse to be accurate only when all query answers are bounded away from T. This is what the known accuracy claim in the literature proves.

The first contribution of this paper is a more general notion of accuracy that takes into account the *distance to disagreement* w.r.t. the "ideal" algorithm that is necessary in accuracy claims for differential privacy algorithms. Informally, an input u of a deterministic computation f has distance to disagreement α (with respect to a distance function d on the input space) if α is the largest number such that whenever $f(u) \neq f(v)$ then $d(u, v) \geq \alpha$. This notion of distance to disagreement is inspired from the Propose-Test-Release mechanism [Dwork and Roth 2014], but we use it for the purpose of accuracy rather than privacy. More precisely, our notion of accuracy requires that for every input u whose distance to disagreement is greater than α, the output of the differentially private algorithm be with in a distance of γ from the correct output, with probability at least $1 - \beta$. The traditional accuracy definition, used in the literature on randomized algorithms, is obtained by setting $\alpha = 0$. Our definition captures most *ad hoc* accuracy claims known in the literature for different differential privacy algorithms. Our definition is reminiscent of accuracy definitions that use three parameters in Blum et al. [2013] and Bhaskar et al. [2010]. However, neither of these definitions have a notion like distance to disagreement [1].

We show that the additional degree of flexibility in our definition of accuracy with the introduction of parameter α, can be exploited to improve known accuracy bounds for NumericSparse, a variant of Sparse that returns the noised query answers when they are above a threshold. Specifically, the accuracy bound from Dwork and Roth [2014] translates in our framework to (α, β, α)-accuracy for all α, and for $\beta = \beta_0(\alpha)$ for some function β_0. In contrast, we can prove (α, β, γ)-accuracy for all α, γ and $\beta = \beta_1(\alpha, \gamma)$. Our result is more general and more precise, since $\beta_1(\alpha, \alpha)$ is approximately $\frac{1}{2}\beta_0(\alpha)$ for all values of α.

Deciding Accuracy. Establishing a general definition of accuracy allows us to study the decidability of the problem of checking accuracy. Differential privacy algorithms are typically parametrized by the privacy budget ϵ, where program variables are typically sampled from distributions whose parameters depend on ϵ. Thus, verifying a property for a differential privacy algorithm is to verify an *infinite family* of programs, obtained by instantiating the privacy budget ϵ to different values. We, therefore, have two parametrized verification problems, which we respectively call the *single-input* and *all-inputs* problems. These problems state: given a parametrized program P_ϵ, an interval I and accuracy bounds (α, β, γ) that may depend on ϵ, is P_ϵ (α, β, γ)-accurate at input u (resp. at all inputs) for all possible values of $\epsilon \in I$?

We first show that accuracy is in general undecidable, both for the single-input and all-inputs variants. Therefore, we focus on decidability for some specific class of programs. We follow the approach from Barthe et al. [2020a], where the authors propose a decision procedure for (ϵ, δ)-differential privacy of a non-trivial class of (parametric) programs, DiPWhile, with a finite number of inputs and output variables taking values in a finite domain. Specifically, we carve out a class of programs, called DiPWhile+, whose operational semantics have a clever encoding as a finite state discrete-time Markov chain. Then, we use this encoding to reduce the problem of accuracy to the theory of reals with exponentials. Our class of programs is larger than the class of programs DiPWhile; it supports the use of a finite number of real input and real output variables and permits the use of real variables as means of Laplace distributions when sampling values.

We show that checking accuracy for both *single-input* and *all-inputs* is decidable for DiPWhile+ programs including those with real input and output variables, assuming Schanuel's conjecture. Schanuel's conjecture is a long-standing open problem in transcendental number theory, with deep applications in several areas of mathematics. In our proof, we use a celebrated result of MacIntyre and Wilkie [1996], which shows that Schanuel's conjecture entails decidability of the theory of reals with exponentials. Our decidability proof essentially encodes the *exact* probability of the algorithm yielding an output that is γ away from the correct answer. As we calculate exact probabilities, we do not have to resort to concentration bounds for verifying accuracy.

[1] A more detailed comparison with these definitions can be found in Section 2.

We also identify sufficient conditions under which the *single-input* problem is decidable for DiPWhile+ programs *unconditionally*, i.e., without assuming Schanuel's conjecture. These unconditional decidability results rely on two crucial observations. First, to check accuracy at specified input u, it suffices to set α to the distance to disagreement for u. This is because β decreases when α increases. Secondly, given a DiPWhile+ program P and real number γ, we can often construct a new DiPWhile+ program P^{new} such that P^{new} outputs true on input u if and only if the output produced by P on u is at most γ away from the correct answer. This allows us only to consider the programs that produce outputs from a finite domain. The single-input accuracy problem can then be expressed in McCallum and Weispfenning [2012]'s decidable fragment of the theory of reals with exponentials.

An immediate consequence of our unconditional decidability results is that the *all-inputs* accuracy problem is decidable for programs when inputs and outputs that come from a finite domain. This is essentially the class of programs DiPWhile, for which differential privacy was shown to be decidable in Barthe et al. [2020a]. Our decidability results are summarized in Table 1 on Page 19.

Experimental Evaluation. We adapt the DiPC tool from Barthe et al. [2020a] to verify accuracy bounds at given inputs and evaluate it on many examples from the literature. Our adaptation takes as input a program in DiPWhile+, constructs a sentence in the decidable McCallum and Weispfenning [2012] fragment of the theory of reals with exponentials and calls Mathematica® to see if the sentence is valid. Using the tool, we verified the accuracy of Sparse, NoisyMax, Laplace Mechanism, and NumericSparse at specified inputs. Our tool also found counter-examples for Sparse and SparseVariant, when accuracy claims do not hold. In addition, our tool is able to verify improvements of accuracy bounds for NoisyMax over known accuracy bounds in the literature given in this paper. Finally, we experimentally found better potential accuracy bounds for our examples by running our tool on progressively smaller β values.

2 DEFINITION OF ACCURACY

In the differential privacy model [Dwork et al. 2006], a trusted curator with access to a 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. However, an overriding concern is that, in spite of the noise, responses should still be sufficiently close to the actual answers to ensure the usefulness of any statistics computed on the basis of those responses. This concern suggests a requirement for accuracy, which is the focus of this paper. The definition of accuracy, one of the main contributions of this paper, is presented here.

We start by considering the usual definition used in theoretical computer science [Motwani and Raghavan 1995] to characterize the quality of a randomized algorithm P that approximately computes a function f. Informally, such a definition demands that, for any input x, the output $P(x)$ be "close" to function value $f(x)$ with "high probability". In these cases, "close" and "high probability" are characterized by parameters (say) γ and β. Unfortunately, such a definition is too demanding, and is typically not satisfied by differential privacy algorithms. We illustrate this with the following example.

Example 1. Consider Sparse (also called Sparse Vector Technique or SVT). The problem solved by Sparse is the following: given a threshold T, a database x and a list of queries of length m, output the list of *indices* of the first c queries whose output on x is above T. Since the goal of Sparse is to maintain privacy of the database x, Sparse introduces some small noise to the query answers as well as to T before comparing them, and outputs the result of the "noisy" comparison; the exact pseudocode is given in Figure 2a. Observe that if the answers to queries are very close to T, then

Sparse's answer will not be "close" to the right answer (for non-noisy comparison) since the noisy comparison could give any result. On the other hand, if the query answer is far from T, then the addition of noise is unlikely to change the result of the comparison, and Sparse is likely to give the right answer with high probability (despite the noise). Thus, Sparse's accuracy claims in the literature do not apply to all inputs, but only to those databases and queries whose answers are far away from the threshold T.

This example illustrates that the accuracy claims in differential privacy can only be expected to hold for inputs that are far away from other inputs that *disagree* — in Example 1 above, accuracy claims don't hold when query answers are close to the threshold because these inputs are close to other inputs in which the comparison with the threshold will give the opposite result. This problem motivates our general definition of accuracy, which captures all accuracy claims for several differential privacy algorithms. Before presenting the formalization, we introduce some notation and preliminaries that will be useful.

Notation. We denote the set of real numbers, rational numbers, natural numbers, and integers by $\mathbb{R}, \mathbb{Q}, \mathbb{N}$, and $\mathbb{Z}$, respectively. We also use $\mathbb{R}^{\infty} = \mathbb{R} \cup \{\infty\}, \mathbb{R}^{>0} = \{x \in \mathbb{R} | x > 0\}, \mathbb{R}^{\geq 0} = \{x \in \mathbb{R} | x \geq 0\}$. The Euler constant is denoted by e. For any $m > 0$ and any vector $a = (a_1, ..., a_m) \in \mathbb{Z}^m$ (or, $a \in \mathbb{R}^m$, $a \in \mathbb{Q}^m$, $a \in \mathbb{N}^m$), recall that $||a||_1 = \sum_{1 \leq i \leq m} |a_i|$ and $||a||_{\infty} = \max\{|a_i| \mid 1 \leq i \leq m\}$.

A differential privacy algorithm is typically a probabilistic program whose behavior depends on the privacy budget ϵ. When we choose to highlight this dependency we denote such programs as P_{ϵ}, and when we choose to ignore it, e.g. when ϵ has been fixed to a particular value, we denote them as just P. Since P is a probabilistic program, it defines a randomized function $[[P]]$, i.e., on an input $u \in \mathcal{U}$ (where $\mathcal{U}$ is the set of inputs), $[[P]](u)$ is a distribution on the set of outputs $\mathcal{V}$. We will often abuse notation and use $P(u)$ when we mean $[[P]](u)$, to reduce notational overhead. For a measurable set $S \subseteq \mathcal{V}$, the probability that P outputs a value in S on input u will be denoted as $\mathrm{Prob}(P(u) \in S)$; when S is a singleton set $\{v\}$ we write $\mathrm{Prob}(P(u) = v)$ instead of $\mathrm{Prob}(P(u) \in \{v\})$.

The accuracy of a differential privacy algorithm P is defined with respect to an "ideal" algorithm that defines the function that P is attempting to compute while maintaining privacy. We denote this "ideal" function as $\det(P)$. It is a deterministic function, and so $\det(P) : \mathcal{U} \to \mathcal{V}$.

To define accuracy, we need a measure for when inputs/outputs are close. On inputs, the function measuring closeness does not need to be a metric in the formal sense. We assume $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}^{\infty}$ is a "distance" function defined on $\mathcal{U}$, satisfying the following properties: for all $u, u' \in \mathcal{U}$, $d(u, u') = d(u', u) \geq 0$, $d(u, u) = 0$. For any $X \subseteq \mathcal{U}$ and any $u \in \mathcal{U}$, we let $d(u, X) = \inf_{u' \in X} d(u, u')$. On outputs, we will need the distance function to be dependent on input values. Thus, for every $u \in \mathcal{U}$, we also assume that there is a distance function d'_u defined on $\mathcal{V}$. For any $v \in \mathcal{V}, u \in \mathcal{U}$ and for any $\gamma \in \mathbb{R}^{\geq 0}$, let $B(v, u, \gamma) = \{v' \in \mathcal{V} \mid d'_u(v, v') \leq \gamma\}$. In other words, $B(v, u, \gamma)$ is a ball of radius γ around v defined by the function d'_u. We next introduce a key notion that we call *distance to disagreement*.

Definition 1. For a randomized algorithm P and input $u \in \mathcal{U}$, the *distance to disagreement* of P and u with respect to $\det(P)$ is the minimum of the distance between u and another input u' such that the outputs of $\det(P)$ on u and u' differ. This can defined precisely as

$$\mathrm{dd}(P, u) = d(u, \{\mathcal{U} - \det(P)^{-1}(\det(P)(u))\}).$$

We now have all the components we need to present our definition of accuracy. Intuitively, the definition says that a differential privacy algorithm P is accurate with respect to $\det(P)$ if on all inputs u that have a large distance to disagreement (as measured by parameter α), P's output on u is close (as measured by parameter γ) to $\det(P)(u)$ with high probability (as measured by parameter β). This is formalized below.

Definition 2. Let $\alpha, \beta, \gamma \in \mathbb{R}^{\geq 0}$ such that $\beta \in [0, 1]$. Let P be a differential privacy algorithm on inputs $\mathcal{U}$ and outputs $\mathcal{V}$. P is said to be (α, β, γ)-accurate at input $u \in \mathcal{U}$ if the following condition holds: if $\mathrm{dd}(P, u) > \alpha$ then $\mathrm{Prob}(P(u) \in B(\mathrm{det}(P)(u), u, \gamma)) \geq 1 - \beta$.

We say that P is (α, β, γ)-accurate if for all $u \in \mathcal{U}$, P is (α, β, γ)-accurate at u.

Observe that when $\alpha = 0$, (α, β, γ)-accuracy reduces to the standard definition used to measure the precision of a randomized algorithm that approximately computes a function [Motwani and Raghavan 1995]. All three parameters α, β, and γ play a critical role in capturing the accuracy claims known in the literature. As we will see in Section 3, we show that the Laplace and Exponential Mechanisms are $(0, \beta, \gamma)$-accurate, AboveThreshold and Sparse are $(\alpha, \beta, 0)$-accurate, and NumericSparse is (α, β, γ)-accurate. Finally, observe that as α, γ increase the error probability β decreases.

Comparison with alternative definitions. As previously noted, it is well-known that the usual definition of accuracy from randomized algorithms does not capture desirable notions of accuracy for differentially private computations, and a number of classic papers from the differential privacy literature have proposed generalizations of the usual notion of accuracy with a third parameter. For instance, Blum et al. [2013] introduce a relaxed notion of accuracy in order to study lower bounds; their definition is specialized to mechanisms on databases, and given with respect to a class C of (numerical) queries. Informally, a mechanism A is accurate if for every query Q in the class C and database D, there exists a nearby query Q' such that with high probability the output $Q(D)$ is close to $Q'(D)$. Their definition is of a very different flavour, and is not comparable to ours. Another generalization is given in Bhaskar et al. [2010], for algorithms that compute frequent items. These algorithms take as input a list of items and return a list of most frequent items and frequencies. Their notion of usefulness requires that with high probability the frequencies are close to the true frequency globally, and for each possible list of items. None of these definitions involve a notion like distance to disagreement.

3 EXAMPLES

The definition of accuracy (Definition 2) is general enough to capture all accuracy claims we know of in the literature. It's full generality seems to be needed in order to capture known results. In this section, we illustrate this by looking at various differential privacy mechanisms and their accuracy claims. As a byproduct of this investigation, we also obtain tighter and better bounds for the accuracy of NumericSparse.

3.1 Laplace Mechanism

The Laplace mechanism [Dwork et al. 2006] is the simplest differential privacy algorithm that tries to compute, in a privacy preserving manner, a numerical function $f : \mathcal{U} \to \mathcal{V}$, where $\mathcal{U} = \mathbb{N}^n$ and $\mathcal{V} = \mathbb{R}^k$, where $k > 0$. The algorithm adds noise sampled from the Laplace distribution. Let us begin by defining this distribution.

Definition 3 (Laplace Distribution)**.** Given $\epsilon > 0$ and mean μ, let $\mathrm{Lap}(\epsilon, \mu)$ be the continuous distribution whose probability density function (p.d.f.) is given by

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

$\mathrm{Lap}(\epsilon, \mu)$ is said to be the *Laplace distribution* with mean μ and scale parameter $\frac{1}{\epsilon}$.

It is sometimes useful to also look at the discrete version of the above distribution. Given $\epsilon > 0$ and mean μ, let $\mathrm{DLap}(\epsilon, \mu)$ be the discrete distribution on $\mathbb{Z}$, whose probability mass function (p.m.f.) is

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

$\mathrm{DLap}(\epsilon,\mu)$ is said to be the *discrete Laplace distribution* with mean μ and scale parameter $\frac{1}{\epsilon}$.

On an input $u \in \mathcal{U}$, instead of outputting $f(u)$, the Laplace mechanism ($P_\epsilon^{\mathsf{Lap}}$) outputs the value $f(x) + (Y_1, \ldots Y_k)$, where each Y_i is an independent, identically distributed random variable from $\mathrm{Lap}((\frac{\epsilon}{\Delta f}, 0))$; here Δf is the *sensitivity* of f, which measures how f's output changes as the input changes [Dwork and Roth 2014].

Theorem 3.8 of Dwork and Roth [2014] establishes the following accuracy claim for Laplace.

Theorem 1 (Theorem 3.8 of Dwork and Roth [2014]). *For any $u \in \mathcal{U}$, and $\delta \in (0, 1]$*

$$Pr\left[||f(u) - P_\epsilon^{\mathsf{Lap}}(u)||_\infty \geq \ln\left(\frac{k}{\delta}\right)\left(\frac{\Delta f}{\epsilon}\right)\right] \leq \delta$$

We can see that Theorem 1 can be rephrased as an accuracy claim using our definition. Observe that here $\det(P_\epsilon^{\mathsf{Lap}}) = f$. Let the distance function d on $\mathcal{U}$ be defined by $d(u,u') = 0$ if $u = u'$ and $d(u,u') = 1$ otherwise. For $u \in \mathcal{U}$, let the distance function d'_u on $\mathcal{V}$ be defined by $d'_u(v,v') = ||v - v'||_\infty$. Now, it is easily seen that the above theorem is equivalent to stating that the Laplace mechanism is $(0, \delta, \gamma)$-accurate for all ϵ, γ, where $\delta = ke^{-\frac{\gamma\epsilon}{\Delta f}}$.

3.2 Exponential Mechanism

Consider the input space $\mathcal{U} = \mathbb{N}^n$. Suppose for an input $u \in \mathcal{U}$, our goal is to output a value in a finite set $\mathcal{V}$ that is the "best" output. Of course for this to be a well-defined problem, we need to define what we mean by the "best" output. Let us assume that we are given a utility function $F : \mathcal{U} \times \mathcal{V} \to \mathbb{R}$ that measures the quality of the output. Thus, our goal on input u is to output $\arg\max_{v\in\mathcal{V}} F(u,v)$ [2].

The Exponential mechanism [McSherry and Talwar 2007] ($P_\epsilon^{\mathsf{Exp}}$) solves this problem while guaranteeing privacy by sampling a value in $\mathcal{V}$ based on the *exponential distribution*. This distribution depends on the utility function F and is defined below.

Definition 4 (Exponential Distribution). Given $\epsilon > 0$ and $u \in \mathcal{U}$, the discrete distribution $\mathrm{Exp}(\epsilon, F, u)$ on $\mathcal{V}$ is given by the probability mass function:

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

On an input $u \in \mathcal{U}$, the Exponential mechanism outputs $v \in \mathcal{V}$ according to distribution $\mathrm{Exp}(\frac{\epsilon}{\Delta F}, F, u)$, where ΔF is the sensitivity of F. Taking $\det(P_\epsilon^{\mathsf{Exp}})$ to be the function such that $\det(P_\epsilon^{\mathsf{Exp}})(u) = \arg\max_{v\in\mathcal{V}} F(u,v)$, the following claim about the Exponential mechanism is proved in Corollary 3.12 of Dwork and Roth [2014].

Theorem 2 (Corollary 3.12 of Dwork and Roth [2014]). *For any u and any $t > 0$,*

$$Pr\left[F(u, P_\epsilon^{\mathsf{Exp}}(u)) \leq F(u, \det(P_\epsilon^{\mathsf{Exp}})(u)) - \frac{2\Delta F}{\epsilon}(\ln(|\mathcal{V}|) + t)\right] \leq e^{-t}$$

Again we can see Theorem 2 as an accuracy claim by our definition. Let the distance function d on $\mathcal{U}$ be defined by $d(u,u') = 0$ if $u = u'$ and $d(u,u') = 1$ otherwise. For any $u \in \mathcal{U}$, take the distance metric d'_u to be $d'_u(v,v') = |F(u,v) - F(u,v')|$, for $v, v' \in \mathcal{V}$. Theorem 2 can be seen

[2] If there are multiple v that maximize the utility, there is a deterministic criterion that disambiguates.

as saying that, for all ϵ and t, the Exponential mechanism is $(0, \beta, \gamma)$-accurate where $\beta = e^{-t}$, $\gamma = \frac{2\Delta F}{\epsilon}(\ln |\mathcal{V}| + \ln(\frac{1}{\beta}))$.

3.3 NoisyMax

Consider the following problem. Given a sequence $(q_1, q_2, \ldots, q_m)$ of elements (with each $q_i \in \mathbb{R}$), output the smallest index of an element whose value is the maximum in the sequence. The algorithm NoisyMax is a differentially private way to solve this problem. It is shown in Figure 1. Based on the privacy budget ϵ, it independently adds noise distributed according to $\mathsf{Lap}(\frac{\epsilon}{2}, 0)$ and then outputs the index with the maximum value after adding the noise.

```
Input: q[1 : m]
Output: out

NoisyVector ← []
for i ← 1 to m do
    NoisyVector[i] ←
      Lap(ε/2, q[i])
end
out ←
  argmax(NoisyVector)
```

Fig. 1. Algorithm NoisyMax

Let us denote the deterministic function that outputs the index of the maximum value in the sequence $(q_1, \ldots, q_m)$ by $\det(\mathsf{NoisyMax})$. On a given input sequence of length m, suppose i is the index output by NoisyMax and j is the index output by $\det(\mathsf{NoisyMax})$. Theorem 6 of Barthe et al. [2016b] proves that, for any $\beta \in (0, 1]$, $Pr(q_j - q_i < \frac{4}{\epsilon} \ln \frac{m}{\beta}) \geq 1 - \beta$.

There are two different ways we can formulate NoisyMax in our framework. In both approaches $\mathcal{U} = \mathbb{R}^m$. In the first approach, the distance function d on $\mathcal{U}$ is the same as the one given for the Laplace mechanism, i.e., for any $u, u' \in \mathcal{U}$, $d(u, u') = 0$ if $u = u'$ and $d(u, u') = 1$ otherwise. The set $\mathcal{V} = \{i : 1 \leq i \leq m\}$. For any $u = (q_1, ..., q_m) \in \mathcal{U}$, the distance function d'_u on $\mathcal{V}$ is defined by $d'_u(i, j) = |q_i - q_j|$. Now, it is easy to see that the above mentioned result of Barthe et al. [2016b], is equivalent to the statement that NoisyMax is $(0, \beta, \gamma)$-accurate where $\beta = me^{-\frac{\gamma\epsilon}{4}}$.

In the second approach, we use the distance functions d on $\mathcal{U}$ and d'_u on $\mathcal{V}$ (for $u \in \mathcal{U}$) defined as follows: for $u, u' \in \mathcal{U}$, $d(u, u') = ||u - u'||_\infty$, and for $u \in \mathcal{U}$, $i, j \in \mathcal{V}$, if $q_i = q_j$ then $d'_u(i, j) = 0$, otherwise $d'_u(i, j) = 1$. We have the following lemma for the accuracy of NoisyMax (See Appendix A).

Lemma 3. *NoisyMax is $(\alpha, \beta, 0)$-accurate for $\beta = me^{-\frac{\alpha\epsilon}{2}}$ and for all $\alpha \geq 0$.*

3.4 AboveThreshold

Given a sequence of queries $(q_1, \ldots, q_m)$ $(q_i \in \mathbb{R})$ and parameter $T \in \mathbb{R}$, consider the problem of determining the first query in the sequence which is *above the threshold* T. The goal is not to output the index of the query, but instead to output a sequence of $\bot$ as long as the queries are below T, and to terminate when either all queries have been read, or when the first query $\geq T$ is read; if such a query is found, the algorithm outputs $\top$ and stops.

AboveThreshold is a differentially private algorithm that solves the above problem. The algorithm is a special case of Sparse shown in Figure 2a when $c = 1$. AboveThreshold works by adding noise to T and to each query, and comparing if the noised queries are below the noised threshold. The noise added is sampled from the Laplace distribution with scale parameter $\frac{2}{\epsilon}$ (for the threshold) and $\frac{4}{\epsilon}$ (for queries). The set of inputs for AboveThreshold is $\mathcal{U} = \mathbb{R}^m$ and the outputs are $\mathcal{V} = \{\bot^m\} \cup \{\bot^k\top \mid k < m\}$. The accuracy claims for AboveThreshold in Dwork and Roth [2014] are given in terms of a notion of (α, β)-*correctness*. Let $\mathcal{A}$ be a randomized algorithm with inputs in $\mathcal{U}$ and outputs in $\mathcal{V}$, and let $\alpha \in \mathbb{R}^{\geq 0}$ and $\beta \in [0, 1]$. We say that $\mathcal{A}$ is (α, β)-correct if for every k, $1 \leq k \leq m$, and for every input $u = (q_1, ..., q_m) \in \mathcal{U}$ such that $q_k \geq T + \alpha$ and $q_i < T - \alpha$ for $1 \leq i < k$, $\mathcal{A}$ outputs $\bot^{k-1}\top$ with probability $\geq 1 - \beta$. Using the results in Dwork and Roth [2014], one can prove the following lemma.

Lemma 4 (Dwork and Roth [2014]). *AboveThreshold is (α, β)-correct for $\beta = 2me^{-\frac{\alpha\epsilon}{8}}$ and for all $\alpha \geq 0$.*

We formulate AboveThreshold in our framework and relate (α, β)-correctness to $(\alpha, \beta, 0)$-accuracy as follows. Let the distance function d on $\mathcal{U}$ be given by $d(u, u') = ||(u - u')||_\infty$. The distance function d'_u on $\mathcal{V}$ is given by $d'_u(v, v') = 0$ if $v = v'$, otherwise $d'_u(v, v') = 1$. Now, we have the following lemma.

Lemma 5. *For any randomized algorithm $\mathcal{A}$ as specified above, for any α, β such that $\alpha \geq 0$ and $\beta \in [0, 1]$, $\mathcal{A}$ is (α, β)-correct iff $\mathcal{A}$ is $(\alpha, \beta, 0)$-accurate.*

Proof. Let α, β be as given in the statement of the lemma. Now, assume $\mathcal{A}$ is (α, β)-correct. We show that it is $(\alpha, \beta, 0)$-accurate. Let $v = \perp^{k-1}\top$ such that $1 \leq k \leq m$. Now consider any $u = (u_1, ..., u_m) \in \det(\mathcal{A})^{-1}(v)$ such that $d(u, \mathcal{U} - (\det(\mathcal{A}))^{-1}(v))) > \alpha$. Now, if $k > 1$ then fix any $i, i < k$ and consider $w = (w_1, ..., w_m) \in \mathcal{U}$ such that $w_i = T$ and $w_j = u_j$ for all $j \leq m$ and $j \neq i$. Clearly $\det(\mathcal{A})(w) \neq v$ and hence $w \in \mathcal{U} - (\det(\mathcal{A}))^{-1}(v)$. Since $d(u, w) > \alpha$, it is the case that $u_i < T - \alpha$. Now, let $\delta \in \mathbb{R}$ such that $\delta > 0$. Consider $w = (w_1, ..., w_m) \in \mathcal{U}$ such that $w_k = T - \delta$ and $w_i = u_i$ for $i \neq k, 1 \leq i \leq m$. Clearly, $w \in \mathcal{U} - (\det(\mathcal{A}))^{-1}(v))$, and hence $d(u, w) > \alpha$. Now, we have $u_k - w_k > \alpha$ and hence $u_k > T - \delta + \alpha$. Since the last inequality holds for any $\delta > 0$, we see that $u_k \geq T + \alpha$. Thus, we see that $u_i < T - \alpha$ for $i < k$ and $u_k \geq T + \alpha$. Since, $\mathcal{A}$ is (α, β)-correct, we see that it outputs v with probability $\geq 1 - \beta$. Hence $\mathcal{A}$ is $(\alpha, \beta, 0)$-accurate.

Now, assume that $\mathcal{A}$ is $(\alpha, \beta, 0)$-accurate. We show that it is (α, β)-correct. Consider any $u = (u_1, ..., u_m) \in \mathcal{U}$ such that, for some $k \leq m$, $u_k \geq T + \alpha$ and for all $i < k$, $u_i < T - \alpha$. As before let $v = \perp^{k-1}\top$. Clearly $\det(\mathcal{A})(u) = v$. Now consider any $w \in \mathcal{U} - (\det(\mathcal{A}))^{-1}(v)$, i.e., $\mathcal{A}(w) \neq v$. It has to be the case that, either for some $i < k$, $w_i \geq T$, or $w_k < T$. In the former case $w_i - u_i > \alpha$ and in the later case, $u_k - w_k > \alpha$. Thus, $d(u, w) > \alpha$ for every $w \in \mathcal{U}$ such that $\det(\mathcal{A})(w) \neq v$. Hence $d(u, \mathcal{U} - (\det(\mathcal{A}))^{-1}(v)) > \alpha$. Since $\mathcal{A}$ is $(\alpha, \beta, 0)$-accurate, it outputs v with probability $\geq 1 - \beta$. Hence $\mathcal{A}$ is (α, β)-correct. □

Using Lemma 5 and Lemma 4, we can conclude that AboveThreshold is $(\alpha, \beta, 0)$-accurate for $\beta = 2me^{-\frac{\alpha\epsilon}{8}}$ and for all $\alpha \geq 0$.

3.5 Sparse

The Sparse algorithm is a generalization of AboveThreshold. As in AboveThreshold, we get a sequence of queries $(q_1, \ldots, q_m)$ and a threshold T, and we output $\perp$ whenever the query is below T, and $\top$ when it is above T. In AboveThreshold the algorithm stops when either the first $\top$ is output or the entire sequence of queries is processed without outputting $\top$. Now, we want to terminate when either c $\top$s are output, or the entire sequence of queries is processed without c $\top$s being output. We will call this the deterministic function det(Sparse), and Sparse is the randomized version of it that preserves privacy. The algorithm Sparse is shown in Figure 2a. The set of inputs $\mathcal{U}$, outputs $\mathcal{V}$, distance metrics d and d'_u are the same as for AboveThreshold (Section 3.4).

Suppose the input sequence of queries $(q_1, ..., q_m)$ satisfies the following property: for all j, $1 \leq j \leq m$, either $q_j < T - \alpha$ or $q_j \geq T + \alpha$, and furthermore, for at most c values of j, $q_j \geq T + \alpha$. A sequence $v = (v_1, ..., v_k) \in \{\perp, \top\}^*$ is a valid output sequence for the above input sequence of queries if $k \leq m$ and the following conditions hold: (i) $\forall j \leq k$, $v_j = \perp$ if $q_j(D) < T - \alpha$, otherwise $v_j = \top$; (ii) if $k < m$ then $v_k = \top$ and there are c occurrences of $\top$ in v; (iii) if $k = m$ and $v_k = \perp$ then there are fewer than c occurrences of $\top$ in v. The following accuracy claim for Sparse can be easily shown by using Theorem 3.26 of Dwork and Roth [2014]. Let $\alpha, \beta \in \mathbb{R}$ be such that $\alpha \geq 0, \beta \in [0, 1]$ and $\alpha = \frac{8c}{\epsilon}(\ln m + \ln(\frac{2c}{\beta}))$. On an input sequence of queries satisfying the above specified property, with probability $\geq (1 - \beta)$, Sparse terminates after outputting a valid output sequence. Now, for

(a) Sparse

```
Input: q[1 : m]
Output: out[1 : m]
r_T ← Lap(ε/2c, T)
count ← 0
for i ← 1 to m do
    r ← Lap(ε/4c, q[i])
    b ← r ≥ r_T
    if b then
        out[i] ← ⊤,
         r_T ← Lap(ε/2c, T)
        count ← count + 1
        if count ≥ c then
            exit
        end
    else
        out[i] ← ⊥
    end
end
```

(b) NumericSparse

```
Input: q[1 : m]
Output: out[1 : m]
r_T ← Lap(4ε/9c, T)
count ← 0
for i ← 1 to m do
    r ← Lap(2ε/9c, q[i])
    b ← r ≥ r_T
    if b then
        out[i] ← Lap(ε/9c, q[i]),
         r_T ← Lap(4ε/9c, T)
        count ← count + 1
        if count ≥ c then
            exit
        end
    else
        out[i] ← ⊥
    end
end
```

Fig. 2. Algorithms Sparse and NumericSparse. SparseVariant (discussed in Section 7.1) is a variant of Sparse where r_T is not re-sampled in the for-loop (shown in bold here).

α, β as given above, using the same reasoning as in the case of the algorithm AboveThreshold, it is easily shown that Sparse is $(\alpha, \beta, 0)$-accurate for $\beta = 2mce^{-\frac{\alpha\epsilon}{8c}}$ and for all $\alpha \geq 0$.

3.6 NumericSparse

Consider a problem very similar to the one that Sparse tries to solve, where we again have a sequence of queries $(q_1, \ldots, q_m)$ and threshold T, but now instead of outputting $\top$ when $q_i \geq T$, we want to output q_i itself. NumericSparse solves this problem while maintaining differential privacy. It is very similar to Sparse and is shown in Figure 2b. The only difference between Sparse and NumericSparse is that instead of outputting $\top$, NumericSparse outputs a q_i with added noise. We now show how accuracy claims about NumericSparse are not only captured by our general definition, but in fact can be improved.

As before, let $\mathcal{U} = \mathbb{R}^m$. The distance function d on $\mathcal{U}$ is defined as follows. For $u, u' \in \mathcal{U}$, where $u = (u_1, ..., u_m)$ and $u' = (u'_1, ..., u'_m)$, if for all i such that both $u_i, u'_i \geq T$, it is the case that $u_i = u'_i$, then $d(u, u') = ||(u - u')||_\infty$, otherwise $d(u, u') = \infty$. We define $\mathcal{V} = (\{\bot\} \cup \mathbb{R})^m$. The distance function d'_u (for any u) on $\mathcal{V}$ is defined as follows: for $v, v' \in \mathcal{V}$, where $v = (v_1, ..., v_m)$ and $v' = (v'_1, ..., v'_m)$, if for all $j, 1 \leq j \leq m$, either $v_j = v'_j = \bot$ or $v_j, v'_j \in \mathbb{R}$ then $d'_u(v, v') = \max\{|v_j - v'_j| \mid v_j, v'_j \in \mathbb{R}\}$, otherwise $d'_u(v, v') = \infty$. Let det(NumericSparse) be the function given by the deterministic algorithm described earlier.

The accuracy for NumericSparse is given by Theorem 3.28 of Dwork and Roth [2014], which can be used to show that it is (α, β, α)-accurate with $\beta = 4mce^{-\frac{\alpha\epsilon}{9c}}$. The following theorem gives a better accuracy result in which β is specified as a function of both α and γ. (See Appendix A for a proof.) In the special case, when $\gamma = \alpha$, we get a value of β (given in Corollary 7) that is smaller than the above value.

Theorem 6. *NumericSparse is (α, β, γ)-accurate where*

$$\beta = 2mce^{-\frac{\alpha\epsilon}{9c}} + ce^{-\frac{\gamma\epsilon}{9c}}$$

Corollary 7. *For any $\alpha > 0$, NumericSparse is (α, β, α)-accurate where $\beta = (2m+1)ce^{-\frac{\alpha\epsilon}{9c}}$*

3.7 SmallDB

The algorithm SmallDB is given in Dwork and Roth [2014]. Using the notation of Dwork and Roth [2014], we let $\mathcal{X}$ denote a finite set of database records. We let $n = |\mathcal{X}|$ and $\mathcal{X} = \{\mathcal{X}_i \mid 1 \leq i \leq n\}$. A database $x = (x_1, \dots, x_n)$ is a n-vector of natural numbers, where x_i denotes the number of occurrences of $\mathcal{X}_i$ in the database. Thus, $\mathbb{N}^n$ is the set of possible databases. The size of the database x is simply $||x||_1$. A query f is a function $f : \mathcal{X} \rightarrow [0, 1]$. The query f is extended to the set of databases, by defining $f(x) = \sum_{1 \leq i \leq n} x_i f(\mathcal{X}_i)$.

The algorithm SmallDB takes as input a database x, a finite set of queries Q and two real parameters $\epsilon, a > 0$ and outputs a small database from a set $\mathcal{R}$, which is the set of small databases of size $\frac{\log |Q|}{\alpha^2}$. Our parameter a is the parameter α of Dwork and Roth [2014]. The algorithm employs the exponential mechanism using the utility function $u : \mathbb{N}^n \times \mathcal{V} \rightarrow \mathbb{R}$, defined by $u(x, v) = -\max\{|f(x) - f(v)| \mid f \in Q\}$. As in the Exponential mechanism, we take $\mathcal{U} = \mathbb{N}^n$ and the distance d to be as in the Laplace mechanism. The distance d'_x is as given in the Exponential mechanism. The function det(SmallDB), is defined exactly as $\det(P_\epsilon^{\mathsf{Exp}})$, i.e., $\det(\mathsf{SmallDB})(x) = \arg\max_v u(x, v)$. For any $x \in \mathcal{U}$, let $v_x = \det(\mathsf{SmallDB})(x)$. Using Proposition 4.4 of Dwork and Roth [2014] and its proof, it can easily be shown that SmallDB is $(0, \beta, \gamma)$-accurate where

$$\gamma = a - |u(x, v_x)| + \frac{2}{\epsilon\, ||\, x||_1} \left(\frac{\log(|X|)\log(|Q|)}{a^2} + \log\left(\frac{1}{\beta}\right) \right)$$

4 THE ACCURACY PROBLEM AND ITS UNDECIDABILITY

Armed with a definition of what it means for a differential privacy algorithm to be accurate, we are ready to define the computational problem(s) associated with checking accuracy claims. Let us fix a differential privacy algorithm P_ϵ and the deterministic algorithm $\det(P_\epsilon)$ against which it will be measured. Let us also fix the set of inputs $\mathcal{U}$ and outputs $\mathcal{V}$ for P_ϵ. Informally, we would like to check if P_ϵ is (α, β, γ)-accurate. Typically, α, β, γ depend on both ϵ and the input u. Furthermore, the program P_ϵ may only be well defined for ϵ belonging to some interval I. Therefore, we introduce some additional parameters to define the problem of checking accuracy.

Let $I \subseteq \mathbb{R}^{\geq 0}$ be an interval with rational end-points. Let $\Sigma = \mathbb{R}^{\geq 0} \times [0, 1] \times \mathbb{R}^{\geq 0}$. Consider $\eta : I \times \mathcal{U} \rightarrow \mathfrak{P}(\Sigma)$, where $\mathfrak{P}(\Sigma)$ denotes the powerset of Σ. Here $\eta(\epsilon, u)$ is the set of all valid (α, β, γ) triples for privacy budget ϵ and input u. η shall henceforth be referred to as the *admissible region.*

Further, let $d : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}^\infty$ be a distance function on $\mathcal{U}$. Let $d' : \mathcal{U} \times \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}^\infty$ be such that for each $u \in \mathcal{U}$, the function $d'_u : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}^\infty$ defined as $d'_u(v_1, v_2) = d'(u, v_1, v_2)$ is a distance function on $\mathcal{V}$. We will call d the *input distance function* and d' the *output distance function.* The following two problems will be of interest.

Accuracy-at-an-input: Given input u, determine if P_ϵ is (α, β, γ)-accurate for all $\epsilon \in I$ and $(\alpha, \beta, \gamma) \in \eta(\epsilon, u)$.

Accuracy-at-all-inputs: For all inputs u, determine if P_ϵ is (α, β, γ)-accurate for all $\epsilon \in I$ and $(\alpha, \beta, \gamma) \in \eta(\epsilon, u)$.

Counterexamples. A counterexample for the accuracy-at-an-input decision problem, with program P_ϵ and $u \in \mathcal{U}$, is a quadruple $(\epsilon_0, \alpha, \beta, \gamma)$ such that $(\alpha, \beta, \gamma) \in \eta(\epsilon_0, u)$ and the program P_{ϵ_0} is not (α, β, γ)-accurate at input u, where η is the function specifying valid sets of parameters as described above. Along the same lines, a counterexample for the accuracy-at-all-inputs decision problem is a quintuple $(u, \epsilon_0, \alpha, \beta, \gamma)$ such that $(\epsilon_0, \alpha, \beta, \gamma)$ is a counterexample for the accuracy-at-an-input decision problem at input u for P_ϵ.

The following theorem shows that the two problems above are undecidable for general randomized programs P_ϵ (See Appendix B for the proof).

Theorem 8. *Both the problems Accuracy-at-an-input and Accuracy-at-all-inputs are undecidable for the general class of randomized programs.*

Remark. For simplicity of presentation, we will assume that the interval I is always the set of strictly positive integers.

5 A DECIDABLE CLASS OF PROGRAMS

In this section we will identify a class of programs, called DiPWhile+, for which we will prove decidability results in Section 6. DiPWhile+ programs are probabilistic while programs that are an extension of the language DiPWhile introduced by Barthe et al. [2020a], for which checking differential privacy was shown to be decidable. Our decidability results rely crucially on the observation that the semantics of DiPWhile+ programs can be defined using finite state parametric DTMCs, whose transition probabilities are definable in first order logic over reals. Therefore, we begin by identifying fragments of first order logic over reals that are relevant for this paper (Section 5.1) before presenting the syntax (Section 5.2) followed by the DTMC semantics (Section 5.3) for DiPWhile+ programs.

5.1 Theory of Reals

Our approach to deciding accuracy relies on reducing the problem to that of checking if a first order sentence holds on the reals. The use of distributions like Laplace and Exponential in algorithms, ensure that the sentences constructed by our reduction involve exponentials. Therefore, we need to consider the full first order theory of reals with exponentials and its sub-fragments.

Recall that $\mathfrak{R}_+ = \langle \mathbb{R}, 0, 1, +, < \rangle$ is the first order structure of reals, with constants 0, 1, addition, and the usual ordering $<$ on reals. The set of first order sentences that hold in this structure, denoted Th_+, is sometimes called the first order theory of linear arithmetic. The structure $\mathfrak{R}_{+,\times} = \langle \mathbb{R}, 0, 1, +, \times, < \rangle$ also has multiplication, and we will denote its first order theory, the theory of real closed fields, as $\mathsf{Th}_{+,\times}$. The celebrated result due to Tarski [1951] is that Th_+ and $\mathsf{Th}_{+,\times}$ admit quantifier elimination and are decidable.

Definition 5. A partial function $f : \mathbb{R}^n \hookrightarrow \mathbb{R}^k$ is said to be *definable* in $\mathsf{Th}_+/\mathsf{Th}_{+,\times}$ respectively, if there are formulas $\psi_f(\bar{x})$ and $\varphi_f(\bar{x}, \bar{y})$ over the signature of $\mathfrak{R}_+$/ $\mathfrak{R}_{+,\times}$ respectively with n and $n + k$ free variables respectively ($\bar{x}$ and $\bar{y}$ are vectors of n and k variables respectively) such that

(1) for all $\bar{a} \in \mathbb{R}^n$, $\bar{a} \in \mathsf{dom}(f)$ iff $\mathfrak{R} \models \psi_f[\bar{x} \mapsto \bar{a}]$, and
(2) for all $\bar{a} \in \mathbb{R}^n$ such that $\bar{a} \in \mathsf{dom}(f)$, $f(\bar{a}) = \bar{b}$ iff $\mathfrak{R} \models \varphi_f[\bar{x} \mapsto \bar{a}, \bar{y} \mapsto \bar{b}]$

where $\mathfrak{R}$ is $\mathfrak{R}_+/\mathfrak{R}_{+,\times}$ respectively.

Finally, the real exponential field $\mathfrak{R}_{\mathsf{exp}} = \langle \mathbb{R}, 0, 1, e^{(\cdot)}, +, \times \rangle$ is the structure that additionally has the unary exponential function $e^{(\cdot)}$ which maps $x \mapsto e^x$. A long-standing open problem in mathematics is whether its first-order theory (denoted here by $\mathsf{Th}^{\mathsf{full}}_{\mathsf{exp}}$) is decidable. However, it was shown by MacIntyre and Wilkie [1996] that $\mathsf{Th}^{\mathrm{full}}_{\mathrm{exp}}$ is decidable provided Schanuel's conjecture (see Lang [1966]) holds for the set of reals[3].

Some fragments of $\mathsf{Th}^{\mathrm{full}}_{\mathrm{exp}}$ with the exponential function are known to be decidable. In particular, there is a fragment identified by McCallum and Weispfenning [2012] that we exploit in our results. Let the language $\mathcal{L}_{\mathrm{exp}}$ be the first-order formulas over a restricted vocabulary and syntax defined as follows. Formulas in $\mathcal{L}_{\mathrm{exp}}$ are over the signature of $\mathfrak{R}_{\mathrm{exp}}$, built using variables $\{\epsilon\} \cup \{x_i \mid i \in \mathbb{N}\}$. In $\mathcal{L}_{\mathrm{exp}}$, the unary function $e^{(\cdot)}$ shall only be applied to the variable ϵ and rational multiples of ϵ. Thus, terms in $\mathcal{L}_{\mathrm{exp}}$ are polynomials with rational coefficients over the variables $\{\epsilon\} \cup \{x_i \mid i \in \mathbb{N}\} \cup \{e^{\epsilon}\} \cup \{e^{q\epsilon} \mid q \in \mathbb{Q}\}$. Atomic formulas in the language are of the form $t = 0$, $t < 0$, or $0 < t$, where t is a term. Quantifier free formulas are Boolean combinations of atomic formulas. Sentences in $\mathcal{L}_{\mathrm{exp}}$ 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 [4]. 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 $\mathsf{Th}_{\mathrm{exp}}$ is the collection of all sentences in $\mathcal{L}_{\mathrm{exp}}$ that are valid in the structure $\mathfrak{R}_{\mathrm{exp}}$. The crucial property for this theory is that it is decidable.

Theorem 9 (McCallum and Weispfenning [2012]). *$\mathsf{Th}_{\mathrm{exp}}$ is decidable.*

We will denote the set of formulas of the form $Q_j x_j \cdots Q_n x_n \psi(\epsilon, x_1, \ldots, x_n)$ by $\mathcal{L}_{\mathrm{exp}}(\epsilon, x_1 \ldots, x_{j-1})$. Finally, our tractable restrictions (and our proofs of decidability) shall often utilize the notion of partial functions being *parametrically definable* in $\mathsf{Th}_{\mathrm{exp}}$; we therefore conclude this section with the formal definition.

Definition 6. A partial function $f : \mathbb{R}^n \times (0, \infty) \hookrightarrow \mathbb{R}^k$ is said to be *parametrically definable* in $\mathsf{Th}_{\mathrm{exp}}$, if there are formulas $\psi_f(\bar{x}, \epsilon)$ and $\varphi_f(\bar{x}, \epsilon, \bar{y})$ over the signature of $\mathfrak{R}_{\mathrm{exp}}$ with $n+1$ and $n+k+1$ free variables respectively ($\bar{x}$ and $\bar{y}$ are vectors of n and k variables respectively, and ϵ is a variable) such that

(1) for every $\bar{a} \in \mathbb{Q}^n$, the formulas $\psi_f[\bar{x} \mapsto \bar{a}]$ and $\varphi_f[\bar{x} \mapsto \bar{a}]$ are in $\mathcal{L}_{\mathrm{exp}}(\epsilon, \bar{y})$,
(2) for all $\bar{a} \in \mathbb{R}^n$, $b \in (0, \infty)$. $(\bar{a}, b) \in \mathrm{dom}(f)$ iff $\mathfrak{R}_{\mathrm{exp}} \models \psi_f[\bar{x} \mapsto \bar{a}, \epsilon \mapsto b]$, and
(3) for all $\bar{a} \in \mathbb{R}^n$, $b \in (0, \infty)$ such that $(\bar{a}, b) \in \mathrm{dom}(f)$, $f(\bar{a}, b) = \bar{c}$ iff $\mathfrak{R}_{\mathrm{exp}} \models \varphi_f[\bar{x} \mapsto \bar{a}, \epsilon \mapsto b, \bar{y} \mapsto \bar{c}]$.

When $n = 0$, we simply say that f is *definable* in $\mathsf{Th}_{\mathrm{exp}}$.

Example 2. Consider the NumericSparse algorithm from Section 3.6. Assume that the algorithm is run on an array of size 2 with threshold T set to 0. Assume that the array elements take the value x_1 and x_2 with $x_1 < 0 < x_2$. Given $x_\gamma \geq 0$, let $p(x_1, x_2, x_\gamma, \epsilon)$ denote the probability of obtaining the output $(\perp, z)$ such that $|x_2 - z| < x_\gamma$. Note that p can be viewed as a partial function $p : R^3 \times (0, \infty) \hookrightarrow R$ with domain $\{(x_1, x_2, x_\gamma, \epsilon) \mid x_1 < 0 < x_2, x_\gamma > 0, \epsilon > 0\}$. Further, $p(x_1, x_2, x_\gamma, \epsilon)$ is the product $p_1 p_2$ where

$$p_1 = 1 - \frac{2}{3}(e^{\frac{2}{9}x_1\epsilon} + e^{-\frac{2}{9}x_2\epsilon}) + \frac{1}{6}(e^{\frac{4}{9}x_1\epsilon} + e^{-\frac{4}{9}x_2\epsilon}) - \frac{1}{48}(e^{\frac{1}{9}(6x_1-2x_2)\epsilon} + e^{-\frac{1}{9}(6x_2-2x_1)\epsilon}) + \frac{1}{4}e^{\frac{2}{9}(x_1-x_2)\epsilon}$$

and $p_2 = 1 - e^{-\frac{1}{9}x_\gamma\epsilon}$.

---

[3]Schanuel's conjecture for reals states that if the real numbers $r_1, \ldots, r_n$ are linearly independent over $\mathbb{Q}$ (the rationals) then the transcendence degree of the field extension $\mathbb{Q}(r_1, \ldots, r_n, e^{r_1}, \ldots, e^{r_n})$ is $\geq n$ (over $\mathbb{Q}$).

[4]Strictly speaking, McCallum and Weispfenning [2012] allow $e^{(\cdot)}$ to be applied only to ϵ. However, any sentence in $\mathcal{L}_{\mathrm{exp}}$ with terms of the form $e^{q\epsilon}$ with $q \in \mathbb{Q}$ can be easily shown to be equivalent to a formula where $e^{(\cdot)}$ is applied only to ϵ. See Barthe et al. [2020b] for examples.

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

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

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

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

Program Statements ($\ell \in$ Labels)

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

Fig. 3. BNF grammar for DiPWhile+. DOM is a finite discrete domain. $\mathcal{F}_{Bool}$, ($\mathcal{F}_{\mathsf{DOM}}$ 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. In addition, DiPWhile+ programs have the restriction that assignments to real and integer variables do not occur with the scope of a while statement.

p is definable in $\mathsf{Th}^{\mathsf{full}}_{\mathsf{exp}}$ with $\psi_p(x_1, x_2, x_\gamma, \epsilon)$ as the formula $(x_1 < 0) \wedge (x_2 > 0) \wedge (x_\gamma > 0) \wedge (\epsilon > 0)$ and $\varphi_p(x_1, x_2, x_\gamma, \epsilon, y)$ as the formula $y = p_1 p_2$. Observe that for rational q_1, q_2, c, the formulas $\psi_p(q_1, q_2, c, \epsilon)$ and $\varphi_p(q_1, q_2, c, \epsilon, y)$ are in $\mathcal{L}_{\mathsf{exp}}(\epsilon, y)$. Hence p is parametrically definable.

Observe that $p(x_1, x_2, x_\gamma, \epsilon)$ is the probability of obtaining an output from NumericSparse that is at most x_γ away from the output of the det(NumericSparse) for inputs of the form $x_1 < 0 < x_2$. This probability is parametrically definable. Such an observation will be true for DiPWhile+ programs and is a crucial ingredient in our decidability results.

5.2 DiPWhile+ **Programs**

Recently, Barthe et al. [2020a,b] identified a class of probabilistic while programs called DiPWhile, for which the problem of checking if a program is differentially private is decidable. Moreover, the language is powerful enough to be able to describe several differential privacy algorithms in the literature that have finite inputs and outputs. In this paper, we extend the language slightly and prove decidability and conditional decidability results for checking accuracy (Section 6). Our extension allows for programs to have real-valued inputs and outputs (DiPWhile programs only have finite-valued inputs) and for these input variables to serve as means of the Laplace mechanisms used during sampling; DiPWhile programs could only use DOM expressions as means of Laplace mechanisms. The resulting class of programs, that we call DiPWhile+, is described in this section.

The formal syntax of DiPWhile+ programs is shown in Figure 3. Program variables can have one of four types: *Bool* ({true, false}); DOM, a finite domain, which is assumed without loss of generality to be a finite subset of integers $\{-N_{\max}, \dots 0, 1, \dots N_{\max}\}$ [5]; integers $\mathbb{Z}$; and reals $\mathbb{R}$. In Figure 3, Boolean/DOM/integer/real program variables are denoted by $\mathcal{B}/\mathcal{X}/\mathcal{Z}/\mathcal{R}$, respectively, and Boolean/DOM/integer/real expressions are given by non-terminals $B/E/Z/R$. 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 the 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. One important restriction to note is that integer-valued expressions cannot be added or multiplied in real-valued expressions.

A DiPWhile+ program 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 3). Private input and public output variables can either be of type DOM or $\mathbb{R}$; this is an important change from DiPWhile where these variables were restricted to be of type DOM. Thus, the set of possible inputs/outputs ($\mathcal{U}/\mathcal{V}$), is identified with the set of valuations for input/output variables. Note that if we represent the set of relevant variables X' as a sequence $\mathsf{x}_1, \mathsf{x}_2, \dots, \mathsf{x}_m$, then a valuation val over X' can be viewed as a sequence $val(\mathsf{x}_1), val(\mathsf{x}_2), \dots, val(\mathsf{x}_m)$.

Program statements are assumed to be uniquely labeled from a set of labels Labels. However, we will often omit these labels, unless they are needed to explain something. 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 expressions or samples drawn using the Laplace or discrete Laplace mechanism. An important distinction to note between programs in DiPWhile+ and DiPWhile by Barthe et al. [2020a], is that when sampling using Laplace, real variables in addition to DOM expressions can be used as the mean. DOM variables are either assigned values of DOM expressions or sampled values. Sampled values for DOM variables can either be drawn using an exponential mechanism ($\mathsf{Exp}(a\epsilon, F(\tilde{\mathsf{x}}), E)$) with a rational-valued, computable scoring function F, or a user-defined distribution ($\mathsf{choose}(a\epsilon, \tilde{E})$), where the probability of picking a value d as function of ϵ according to choose is parametrically definable in $\mathsf{Th}_{\exp}$ as a function of ϵ. Moreover, we assume that there is an algorithm that on input $a, \tilde{d}$ returns the formula defining the probability of sampling $d \in$ DOM from the distribution $\mathsf{choose}(a\epsilon, \tilde{d})$, where $\tilde{d}$ is a sequence of values from DOM. For assignments to Boolean variables, it is worth directing attention to the cases where a variable is assigned the result of comparing two expressions. Notice that the syntax does not allow comparing real and integer expressions. This is an important restriction to get decidability. For technical convenience, we assume that in any execution, variables appearing on the right side of an assignment are assigned a value earlier in the execution.

In addition to the syntactic restrictions given by the BNF grammar in Figure 3, we require that DiPWhile+ programs satisfy the following restriction; this restriction is also used in defining DiPWhile by Barthe et al. [2020a].

Bounded Assignments Real and integer variables are not assigned within the scope of a while loop. Therefore, real and integer variables are assigned only a *bounded* number of times in any execution. Thus, without loss of generality, we assume that real and integer variables are

[5]The distinction between Booleans and finite domain types is for convenience rather than technical neccessity. Moreover, DOM can be any finite set, including a subset of rationals.

assigned *at most once*, as a program with multiple assignments to a real/integer variable can always be rewritten to an equivalent program where each assignment is to a fresh variable.

DiPWhile **and** DiPWhile+**.** DiPWhile, introduced by Barthe et al. [2020a], is a rich language that can describe differential privacy mechanisms with finitely many input variables taking values over a finite domain, and output results over a finite domain. Programs can sample from continuous and discrete versions of Laplacian distributions, user-defined distributions over DOM and exponential mechanism distributions with finite support. Any DiPWhile program can be rewritten as a program in which variables are initially sampled from Laplacian distributions, comparisons between linear combinations of sampled values and inputs are stored in Boolean variables, followed by steps of a simple probabilistic program with Boolean and DOM variables. DiPWhile can express several differential privacy mechanisms such as the algorithms AboveThreshold, NoisyMax, Sparse and exponential mechanism discussed in Section 3. Other examples expressible in DiPWhile include private vertex cover [Gupta et al. 2010] and randomized response. It can also, for example, express versions of NoisyMax where the noise is sampled from an exponential distribution and not from a Laplacian distribution. In DiPWhile+, we allow inputs to take real values. Further, we allow programs to output real values formed by linear combinations of input and sampled real variables. This allows us to express mechanisms such as the private smart sum algorithm [Chan et al. 2011] and NumericSparse (See Section 3.6). In DiPWhile, we could only approximate these examples by discretizing the output values and restricting the input variables to take values in DOM. DiPWhile+ does not allow using Gaussian mechanisms to sample, primarily because our decision procedures do not extend to such algorithms.

Example 3. Algorithm 1 shows how NumericSparse can be encoded in our language with $T = 0, \delta = 0, N = 2, c = 1$; this is a specialized version of the pseudocode in Figure 2b. The algorithm either outputs $\perp$ or a numeric value. We don't have variables of such a type in our language. We therefore encode each output as a pair: DOM variable o^1 and real variable o^2. If $\mathsf{o}^1 = 0$ then output is $\perp$ and if $\mathsf{o}^1 = 1$ then the output is o^2. Though for-loops are not part of our program syntax, they can modeled as while loops, or if bounded (as they are here), they can be unrolled.

5.3 Semantics

A natural semantics for DiPWhile+ programs can be given using Markov kernels. Given a fixed $\epsilon > 0$, the states in such a semantics for 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 $h_{Bool}, h_{\mathsf{DOM}}, h_{\mathbb{Z}}$, and $h_{\mathbb{R}}$ assign values to the Boolean, DOM, real, and integer variables of the program P_ϵ. There is a natural σ-algebra that can be defined on such states, and the semantics defines a Markov kernel over this algebra. Such a semantics for DiPWhile+ would be similar to the one for DiPWhile given by Barthe et al. [2020b], and is skipped here. Throughout the paper, we shall also assume that DiPWhile+ programs terminate with probability 1 on all inputs.

Our decidability results rely crucially on the observation that the semantics of DiPWhile+ can be defined using a *finite-state* (parametrized) DTMC. This semantics, though not natural, can be shown to be equivalent to the Markov kernel semantics. The proof of equivalence is similar to the one given by Barthe et al. [2020b]. We spend the rest of this section highlighting the main aspects of the DTMC semantics that help us underscore the ideas behind our decision procedure. We begin by recalling the definition of a finite-state parametrized DTMC.

Definition 7. A *parametrized DTMC* over $(n+1)$ parameters $(\overline{x}, \epsilon)$ is a pair $\mathcal{D} = (Z, \Delta)$, where Z is a finite set of states, and $\Delta : Z \times Z \to (\mathbb{R}^n \times (0, \infty) \to [0, 1])$ is the *probabilistic transition function*. For any pair of states z, z', $\Delta(z, z')$ will be called the *probability of transitioning* from z to

```
    Input: q1, q2
    Output: o_1^1, o_1^2, o_2^1, o_2^2

 1  T ← 0;
 2  o_1^1 ← 0;
 3  o_2^1 ← 0;
 4  r_T ← Lap(4ε/9, T);
 5  r_1 ← Lap(2ε/9, q1);
 6  b ← r_1 ≥ r_T;
 7  if b then
 8  |   o_1^1 ← 1
 9  |   o_1^2 ← Lap(ε/9, q1)
    else
10  |   r_2 ← Lap(2ε/9, q2);
11  |   b ← r_2 ≥ r_T;
12  |   if b then
13  |   |   o_2^1 ← 1
14  |   |   o_2^2 ← Lap(ε/9, q2)
    |   end
    end
15  exit
```

Algorithm 1: NumericSparse with $N = 2$, $c = 1$, $\delta = 0$, and $T = 0$. The numbers at the beginning of a line indicate the label of the statement.

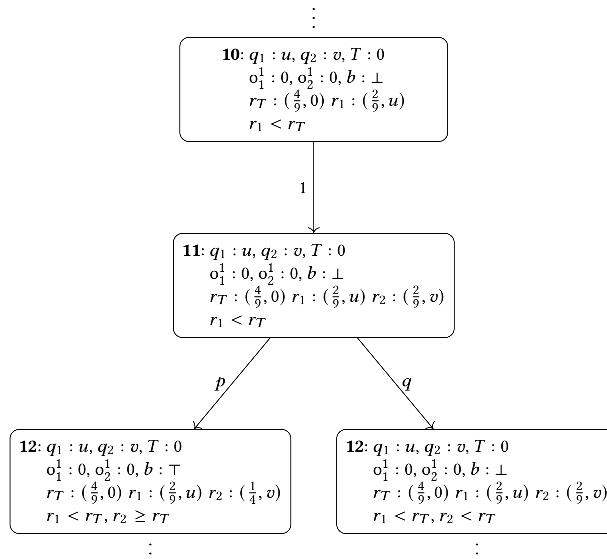

Fig. 4. Partial DTMC semantics of Algorithm 1 showing the steps when lines 10 and 11 are executed. q_1 and q_2 are assumed to have values u and v, respectively. Only values of assigned program variables are shown. Third line in state shows parameters for the real values that were sampled. Last line shows the accumulated set of Boolean conditions that hold on the path.

z', and is a function that, given $\overline{a} \in \mathbb{R}^n$ and $b \in (0, \infty)$, returns a real number between 0 and 1, such that for any state z, $\sum_{z' \in Z} \Delta(\overline{a}, b)(z, z') = 1$.

The connection between programs in DiPWhile+ and parametrized DTMCs is captured by the following result that is exploited in our decidability results.

Theorem 10. *Let P_ϵ be an arbitrary* DiPWhile+ *program whose real-valued input variables are $\overline{x}$. There is a finite state parametrized DTMC $[[P_\epsilon]]$ over parameters $(\overline{x}, \epsilon)$ (with transition function Δ) that is equivalent to the Markov kernel semantics of P_ϵ. Further, the DTMC $[[P_\epsilon]]$ is effectively constructible, and for any pair of states z, z', the partial function $\Delta(z, z')$ is parametrically definable in* $\mathsf{Th}_{\mathsf{exp}}$.

Proof Sketch. The formal construction of the parametrized DTMC $[[P_\epsilon]]$ is very similar the one outlined in Barthe et al. [2020b] for (the restricted) DiPWhile programs. Here, we just sketch the main ideas. It is useful to observe that defining a *finite-state* semantics for DiPWhile+ programs is not obvious, since these programs have real and integer valued variables. The key to obtaining such a finite state semantics is to not track the values of real and integer variables explicitly, but rather implicitly through the relationships they have amongst each other.

Informally, a state in $[[P_\epsilon]]$ keeps track of a program statement to be executed (in terms of its label), and the values stored in each of the Boolean and DOM variables. However, the values of real and integer variables will not be explicitly stored in the state. Recall that real and integer variables are assigned a value only once in a DiPWhile+ program. Therefore, states of $[[P_\epsilon]]$ store the (symbolic) expression on the right side of an assignment for each real/integer variable, instead of the actual value; when the value is sampled, the symbolic parameters of the distribution are stored. In addition to symbolic values for real and integer variables, a state of $[[P_\epsilon]]$ also tracks the relative order among the values of real and integer variables. Thus, $[[P_\epsilon]]$ has only finitely many states. A state of $[[P_\epsilon]]$ is an abstraction of all "concrete states" whose assignments to Boolean and DOM variables match, and whose assignment to real and integer variables satisfy the constraints imposed by the symbolic expressions and the relative order maintained in the $[[P_\epsilon]]$ state.

State updates in $[[P_\epsilon]]$ are as follows. Assignments to DOM variables are as expected — a new value is calculated and stored in the state for a deterministic assignment, or a value is sampled probabilistically and stored in the state for a randomized assignment. Assignments to real and integers variables are *always* deterministic — the state is updated with the appropriate symbolic values that appear in the deterministic or probabilistic assignment. It is important to note that sampling a value using a Laplace mechanism is a deterministic step in the DTMC semantics. Assignments to Boolean variables, where the right hand side is a Boolean expression, is as expected; the right hand side expression is evaluated and the state is updated with the new value. Assignments to Boolean variables by comparing two real or integer expressions is handled in a special way. These are *probabilistic transitions*. Consider an assignment $\mathrm{b} \leftarrow R_1 \sim R_2$ for example. The result of executing this statement from state z will move to a state where $R_1 \sim R_2$ is added to the set of ordering constraints, with probability equal to the probability that $R_1 \sim R_2$ holds conditioned on the ordering constraints in z holding, subject to the variables being sampled according to the parameters stored in z. With the remaining probability, $[[P_\epsilon]]$ will move to a state where $\neg(R_1 \sim R_2)$ is added to the ordering constraints. Finally, branching and while statements are deterministic steps with the next state being determined by the value stored for the Boolean variable in the condition.

Notice here that since input variables and the privacy parameter ϵ can appear as parameters of the Laplace/discrete Laplace mechanism used to sample a value of a real/integer variable, the transition probabilities of $[[P_\epsilon]]$ depend on these parameters. That these transition probabilities are parametrically definable in $\mathsf{Th}_{\mathsf{exp}}$ can be established along the same lines as the proof that the transition probabilities are definable in $\mathsf{Th}_{\mathsf{exp}}$ for the DTMC semantics of DiPWhile. □

Example 4. The parametrized DTMC semantics of Algorithm 1 is partially shown in Figure 4. We show only the transitions corresponding to executing lines 10 and 11 of the algorithm, when $q_1 = u$ and $q_2 = v$ initially; here $u, v \in \{\perp, \top\}$. 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 has the Boolean conditions that hold along a path. Since 10 and 11 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 11 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 $r_1 < r_T$.

6 DECIDING ACCURACY FOR DIPWHILE+ PROGRAMS

We shall now show that the problem of checking the accuracy of a DiPWhile+ program is decidable, assuming Schanuel's conjecture. Further, we shall identify special instances under which the problem of checking the accuracy of a DiPWhile+ program is decidable without assuming Schanuel's conjecture. Our results are summarized in Table 1.

Remark. For the rest of the section, we shall say that inputs/outputs to the DiPWhile+ program P_ϵ are rational if all the real variables in the input/output respectively take rational values. We shall also say that P_ϵ has finite inputs if all of its input variables are DOM-variables, and that P_ϵ has finite outputs if all of its output variables are DOM-variables.

Result	Problem	Schanuel	Infinite Inputs	Infinite Outputs	$\det(P_\epsilon)$	d	d'	Region η
Thm 12	all-inputs	✓	✓	✓	$\mathsf{Th}_{+,\times}$	$\mathsf{Th}_{+,\times}$	Th_{+}	param. def. in $\mathsf{Th}_{\exp}$
Cor 17	all-inputs	-	✗	✗	Th_{+}	Th_{+}	Th_{+}	(α,γ)-monotonic
Cor 13	an-input	✓	-	✓	$\mathsf{Th}_{+,\times}$	$\mathsf{Th}_{+,\times}$	Th_{+}	param. def. in $\mathsf{Th}_{\exp}$
Thm 14	an-input	-	-	✓	Th_{+}	$\mathsf{Th}_{+,\times}$	Th_{+}	simple, fixed α, fixed γ
Thm 15	an-input	-	-	✓	Th_{+}	Th_{+}	Th_{+}	limit-def., fixed γ
Thm 16	an-input	-	-	✗	Th_{+}	Th_{+}	Th_{+}	(α,γ)-monotonic

Table 1. Summary of our decidability results. The column, Schanuel, indicates whether the result is conditional on Schanuel's conjecture. The column, Problem, indicates if the decision problem is Accuracy-at-all-inputs or Accuracy-at-an-input. The column, Infinite Inputs, indicates if the result allows real variables as inputs. Note that this column is relevant only for the Accuracy-at-all-inputs decision problem. The column, Infinite Outputs, indicates if the result allows real variables as outputs. The columns, $\det(P_\epsilon)$, d and d', indicate the definability assumptions needed for deterministic function $\det(P_\epsilon)$, input distance function d and output distance function d'. The column, Region η indicates the assumptions needed on admissible region.

6.1 Definability assumptions

Let P_ϵ be a DiPWhile+ program with ℓ input DOM-variables, k input real variables, m output DOM-variables, and n output real variables. Observe that DOM is a subset of integers. Hence, the input set $\mathcal{U} = \mathsf{DOM}^\ell \times \mathbb{R}^k$ can be viewed as a subset of $\mathbb{R}^{\ell+k}$, and the output set $\mathcal{V}$ as a subset of $\mathbb{R}^{m+n}$. Thus, $\det(P_\epsilon)$ can be viewed as a partial function from $\mathbb{R}^{\ell+k}$ to $\mathbb{R}^{m+n}$.

Also, observe that $\mathbb{R}^\infty$ can be seen as a subset of $\mathbb{R}\times\mathbb{R}$ by identifying $r \in \mathbb{R}$ with $(0, r)$ and ∞ with $(1, 0)$. Thus, the input distance function, d, can be viewed as a partial function from $\mathbb{R}^{\ell+k} \times \mathbb{R}^{\ell+k}$ to $\mathbb{R}^2$ and the output distance function, d', as a partial function from $\mathbb{R}^{\ell+k} \times \mathbb{R}^{m+n} \times \mathbb{R}^{m+n}$ to $\mathbb{R}^2$. Our results shall require that these functions be definable in sub-theories of real arithmetic.

Finally, recall that the admissible region η is a function that given a privacy budget ϵ and input $\bar{u}$ gives the set of the set of all valid (α, β, γ) for that ϵ and input $\bar{u}$. Observe that η can also be viewed as a function that takes $\alpha, \beta, \gamma, \bar{u}$ and ϵ as input and returns 1 if the triple (α, β, γ) is in the set $\eta(\epsilon, \bar{u})$ and 0 otherwise.

Definition 8. Let P_ϵ be a DiPWhile+ program implementing the deterministic function $\det(P_\epsilon)$. Let d be a distance function on the set of inputs of P_ϵ and d' be the input-indexed distance function on the set of outputs of P_ϵ. Let P_ϵ have ℓ input DOM-variables, k input real variables, m output DOM-variables, and n real output variables. Let η denote the admissible region.

- $\det(P_\epsilon)$ is said to be definable in $\mathsf{Th}_{+,\times}$ (Th_{+} respectively) if it is definable in $\mathsf{Th}_{+,\times}$ (Th_{+} respectively) when viewed as a partial function from $\mathbb{R}^{\ell+k}$ to $\mathbb{R}^{m+n}$.
- d is said to be definable in $\mathsf{Th}_{+,\times}$ (Th_{+} respectively) if it is definable in $\mathsf{Th}_{+,\times}$ (Th_{+} respectively) when viewed as a partial function from $\mathbb{R}^{\ell+k} \times \mathbb{R}^{\ell+k}$ to $\mathbb{R}^2$.
- d' is said to be definable in $\mathsf{Th}_{+,\times}$ (Th_{+} respectively) if it is definable in $\mathsf{Th}_{+,\times}$ (Th_{+} respectively) when viewed as a partial function from $\mathbb{R}^{\ell+k} \times \mathbb{R}^{m+n} \times \mathbb{R}^{m+n}$ to $\mathbb{R}^2$.
- The admissible region η is said to be parametrically definable in $\mathsf{Th}_{\exp}$ if the partial function $h : \mathbb{R}^{3+\ell+k} \times (0, \infty) \to \mathbb{R}$ defined as

$$h(x, y, z, \bar{u}, \epsilon) = \begin{cases} 1 & \text{if } \epsilon > 0 \text{ and } (x, y, z) \in \eta(\epsilon, \bar{u}) \\ 0 & \text{otherwise} \end{cases}$$

is parametrically definable in $\mathsf{Th}_{\exp}$.

Now, we could have chosen to write $\mathsf{det}(P_\epsilon)$ in DiPWhile+ by considering programs that do not contain any probabilistic assignments. Please note that a deterministic program written in DiPWhile+ can be defined in Th_+. Intuitively, this is because we do not allow assignments to integer and real random variables inside loops of DiPWhile+ programs. Hence the loops can be "unrolled". This means that there are only finitely many possible executions of a deterministic DiPWhile+ program, and these executions have finite length. The Boolean checks in the program determine which execution occurs on an input, and these checks can be encoded as formulas in linear arithmetic.

6.2 Decidability assuming Schanuel's conjecture

We start by establishing the following Lemma, which says that if a DiPWhile+ program has only finite outputs (i.e., only DOM-outputs) then the probability of obtaining an output $\bar{v}$ is parametrically definable in $\mathsf{Th}_{\mathsf{exp}}$. The proof of this fact essentially mirrors the proof of the fact that this probability is definable in $\mathsf{Th}_{\mathsf{exp}}$ (without parameters) for the (restricted) DiPWhile programs established in Barthe et al. [2020b]. It based on the observation that it suffices to compute the probability of reaching certain states (labeled exit states) of the DTMC semantics, which have $\bar{v}$ as the valuation over output variables. The reachability probabilities can be computed as a solution to a linear program (with transition probabilities as the coefficients).

Lemma 11. *Let P_ϵ be a* DiPWhile+ *program with finite outputs. Let P_ϵ have ℓ input* DOM*-variables, k input real variables and m output variables. Given, $\bar{v} \in \mathsf{DOM}^m$, let $\mathsf{Pr}_{\bar{v},P_\epsilon} : \mathbb{R}^{\ell+k+1} \hookrightarrow \mathbb{R}$ be the partial function whose domain is $\mathsf{DOM}^\ell \times \mathbb{R}^{k+1}$, and which maps $(\bar{r}, \epsilon)$ to the probability that P_ϵ outputs $\bar{v}$ on input $\bar{r}$. For each $\bar{v}$, the function $\mathsf{Pr}_{\bar{v},P_\epsilon}$ is parametrically definable in $\mathsf{Th}_{\mathsf{exp}}$.*

The following result gives sufficient conditions under which the decision problem Accuracy-at-all-inputs is decidable for DiPWhile+ programs. The conditions state that the deterministic program and the input distance function be definable using first-order theory of real arithmetic. The output distance distance function is required to be definable in the first-order theory of linear arithmetic. Intuitively, this additional constraint is needed as it implies that we only need to compute probabilities that the outputs reside in a region defined by linear equalities and linear inequalities.

Theorem 12. *Assuming Schanuel's conjecture, the problem Accuracy-at-all-inputs is decidable for* DiPWhile+ *programs P_ϵ when (a) $\mathsf{det}(P_\epsilon)$ is definable in $\mathsf{Th}_{+,\times}$, (b) d is definable in $\mathsf{Th}_{+,\times}$, (c) d' is definable in Th_+, and (d) η is parametrically definable in $\mathsf{Th}_{\mathsf{exp}}$.*

The problem Accuracy-at-an-input is also decidable under the same constraints as given by Theorem 12. This is established as a corollary to the proof of Theorem 12 (see Appendix C).

Corollary 13. *Assuming Schanuel's conjecture is true for reals, the problem Accuracy-at-an-input is decidable for* DiPWhile+ *programs P_ϵ and rational inputs $\bar{u}$ when (a) $\mathsf{det}(P_\epsilon)$ is definable in $\mathsf{Th}_{+,\times}$, (b) d is definable in $\mathsf{Th}_{+,\times}$, (c) d' is definable in Th_+, and (d) η is parametrically definable in $\mathsf{Th}_{\mathsf{exp}}$.*

6.3 Unconditional Decidability Results

We shall now give sufficient conditions under which the problems Accuracy-at-an-input and Accuracy-at-all-inputs will be decidable *unconditionally*, i.e., without assuming Schanuel's conjecture. For these results, we will have to restrict the admissible region. All examples discussed in Section 3 have regions that satisfy these restrictions under reasonable assumptions. We start by defining one restriction on regions that will be needed by all our unconditional decidability results. Intuitively, this restriction says that α, γ are independent of the privacy budget, while β is a function of α, γ, ϵ and the input. In addition, we require that β in the region is anti-monotonic in α and γ (condition 5 below).

Definition 9. The admissible region η is simple if there is a partial function $I_{\alpha,\gamma} : \mathbb{R}^p \hookrightarrow \mathfrak{P}(\mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0})$ and a partial function $f_\beta : \mathbb{R}^{2+p} \times (0,\infty) \hookrightarrow [0,1]$ such that

(1) $domain(I_{\alpha,\gamma}) = \mathcal{U}$ where $\mathcal{U} \subseteq \mathbb{R}^p$,
(2) $domain(f_\beta) = \{(a,c,\bar{u},\epsilon) \mid \bar{u} \in \mathcal{U}, (a,c) \in I_{\alpha,\gamma}(\bar{u}), \epsilon > 0)\}$,
(3) f_β is parametrically definable in $\mathsf{Th}_{\exp}$,
(4) $(a,b,c) \in \eta(\epsilon,\bar{u})$ iff $(a,c) \in I_{\alpha,\gamma}(\bar{u}), \bar{u} \in \mathcal{U}$ and $f_\beta(a,c,\bar{u},\epsilon) = b$, and
(5) for all $\epsilon, \bar{u}, a_1, a_2, c_1, c_2$ with $(a_i,c_i) \in I_{\alpha,\gamma}(\bar{u})$ for $i \in \{1,2\}$, and $a_1 \leq a_2, c_1 \leq c_2, f_\beta(a_2,c_2,\bar{u},\epsilon) \leq f_\beta(a_1,c_1,\bar{u},\epsilon)$.

Example 5. Let P_ϵ be a program with $\mathcal{U}$ as the set of inputs. Let η_1 be the region defined as follows. For each $\epsilon > 0$ and input $\bar{u} \in \mathcal{U}$, $\eta_1(\bar{u},\epsilon) = \{(a,b,0) \mid a \geq 0, b = e^{-\frac{a\epsilon}{2}}\}$. η_1 is a simple region with

$$I_{\alpha,\gamma}(\bar{u}) = \begin{cases} \{(a,0) \mid a \geq 0\} & \text{if } \bar{u} \in \mathcal{U} \\ \text{undefined} & \text{otherwise} \end{cases}$$

and

$$f_\beta(a,c,\bar{u},\epsilon) = \begin{cases} e^{-\frac{a\epsilon}{2}} & \text{if } a \in \mathbb{R}^{\geq 0}, c = 0, \bar{u} \in \mathcal{U}, \epsilon > 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Notice that, since $e^{-\frac{a_2\epsilon}{2}} \leq e^{-\frac{a_1\epsilon}{2}}$ if $a_1 \leq a_2$, the region η_1 satisfies condition 5 of Definition 9.

On the other hand, the region η_2 defined as $\eta_2(\bar{u},\epsilon) = \{(a,b,0) \mid a \geq \epsilon, b = e^{-\frac{a\epsilon}{2}}\}$ is not a simple region because α depends on ϵ.

Remark. For the rest of this section, we assume that η is represented by the pair $(I_{\alpha,\gamma}, f_\beta)$. For inputs to decision problems, f_β will be represented by the formulas (ψ_β, ϕ_β) defining it. $I_{\alpha,\gamma}$ will usually represented by a first-order formula $\theta_{\alpha,\gamma}(x_\alpha, x_\gamma, \bar{x})$ such that for all $a, c, \bar{u}$, $(a,c) \in I_{\alpha,\gamma}(\bar{u})$ iff $\theta_{\alpha,\gamma}(a,c,\bar{u})$ is true.

Program with infinite outputs. We start by showing that the problem of checking accuracy for DiPWhile+ programs at a rational input $\bar{u}$ is decidable when we fix α, γ to be some rational numbers. For this result, we shall require that the deterministic function $\det(P_\epsilon(\bar{u}))$ be definable in Th_+. This implies that the output of the function at $\bar{u}$ must be rational. The proof essentially requires that the program $P_\epsilon^{\mathsf{new}}$ constructed in the proof of Theorem 12 be executed on $\bar{u}$, $\det(P_\epsilon(\bar{u}))$ and γ. The assumption that $\det(P_\epsilon(\bar{u}))$ is definable in Th_+ will ensure that the inputs to $P_\epsilon^{\mathsf{new}}$ are rational numbers. Thus, the probability of P_ϵ generating an output on input $\bar{u}$ that is at most γ away, can then be defined in $\mathsf{Th}_{\exp}$. This observation, together with the parametric definability of f_β allows us to show that the sentence constructed in the proof of Corollary 13 that checks accuracy at $\bar{u}$ is a sentence in $\mathcal{L}_{\exp}$. In the decision procedure, we need to provide only the fixed values of α and γ as a description of $I_{\alpha,\gamma}$. The formal proof can be found in Appendix E.

Theorem 14. *The problem Accuracy-at-an-input is decidable for* DiPWhile+ *programs P_ϵ and rational inputs $\bar{u}$ when (a)* $\det(P_\epsilon)$ *is definable in* Th_+*, (b) d is definable in* $\mathsf{Th}_{+,\times}$*, (c) d' is definable in* Th_+*, (d) $\eta = (I_{\alpha,\gamma}, f_\beta)$ such that η is simple and $I_{\alpha,\gamma}(\bar{u}) = \{(a,c)\}$ for some rational numbers a, c.*

One natural question is if the above result can be extended to checking accuracy for varying α, γ. We shall show that, with additional restrictions, we can establish decidability of Accuracy-at-an-input when only γ is fixed. Intuitively, this result will exploit the fact that for a given input $\bar{u}$, the interesting α to consider is the distance to disagreement for $\bar{u}$ (as β decreases with increasing α). We can then proceed as in the proof of Theorem 14. This idea mostly works except that one has to ensure that the distance to disagreement is a rational number to apply Theorem 14; the function f_β does not jump at the point of disagreement; and that we can compute f_β at ∞ (for the case when the distance to disagreement is ∞). The following definition captures the latter two restrictions.

Definition 10. The simple region $\eta = (I_{\alpha,\gamma}, f_\beta)$ is said to be limit-definable if

(1) There is a linear arithmetic formula $\theta_{\alpha,\gamma}(x_\alpha, x_\gamma, \bar{x})$ such that for all $a, c, \bar{u}$, $(a, c) \in I_{\alpha,\gamma}(\bar{u})$ iff $\theta_{\alpha,\gamma}(a, c, \bar{u})$ is true.

(2) There is a partial function $h_\beta : \mathbb{R}^\infty \times \mathbb{R}^\infty \times \mathbb{R}^p \times (0, \infty) \hookrightarrow [0, 1]$ (called f_β's *limit extension*) such that h_β has the following properties.
- *domain*(h_β) is the set of all $(a, c, \bar{u}, \epsilon) \in \mathbb{R}^\infty \times \mathbb{R}^\infty \times \mathbb{R}^p \times (0, \infty)$ such that there is a non-decreasing sequence $\{(a_i, c_i)\}_{i=0}^\infty \in I_{\alpha,\gamma}(\bar{u})$ (i.e., $a_i \leq a_{i+1}, c_i \leq c_{i+1}$) with $\lim_{i\to\infty}(a_i, c_i) = (a, c)$.
- For any non-decreasing sequence $\{(a_i, c_i)\}_{i=0}^\infty \in I_{\alpha,\gamma}(\bar{u})$ such that $\lim_{i\to\infty}(a_i, c_i) = (a, c)$, $h_\beta(a, c, \bar{u}, \epsilon) = \lim_{i\to\infty} f_\beta(a_i, c_i, \bar{u}, \epsilon)$.
- h_β is parametrically definable in $\mathsf{Th}_{\exp}$.

Observe that f_β and its limit extension h_β agree on *domain*(f_β). Therefore, a limit-definable region $\eta = (I_{\alpha,\gamma}, f_\beta)$ shall be represented by a triple $(\theta_{\alpha,\gamma}, \psi_h, \phi_h)$ where (ψ_h, ϕ_h) defines h_β (the limit extension of f_β).

Intuitively, the first requirement ensures that the α that needs to be considered for a fixed γ is rational. The second requirement ensures that f_β is continuous "from below" and can be extended to its boundary (including the case when α takes the value ∞.) Note that ψ_h can be written in the theory of linear arithmetic thanks to the fact that $\theta_{\alpha,\gamma}$ is a linear arithmetic formula.

Example 6. The region η_1 in Example 5 can be seen to limit-definable with $\theta_{\alpha,\gamma}(x_\alpha, x_\gamma, \bar{x}) = ((x_\alpha \geq 0) \wedge (x_\gamma = 0))$ and h_β as follows:

$$h_\beta(a, c, \bar{u}, \epsilon) = \begin{cases} 0 & \text{if } a = \infty, c = 0, \bar{u} \in \mathcal{U}, \epsilon > 0 \\ e^{-\frac{a\epsilon}{2}} & \text{if } a \in \mathbb{R}^{\geq 0}, c = 0, \bar{u} \in \mathcal{U}, \epsilon > 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

An example of a region that is simple but not limit-definable is the region η_3 defined as follows. Given $\bar{u} \in \mathcal{U}$, $\epsilon > 0$, $\eta_3(\bar{u}, \epsilon) = \{(a, b, 0) \mid \text{ either } (0 \leq a < 1 \wedge b = e^{-\frac{a\epsilon}{2}}) \text{ or } (1 \leq a \wedge b = e^{-\frac{3a\epsilon}{5}})\}$. η_3 is not limit-definable as parameter β has a "discontinuity" at $\alpha = 1$.

We have the following theorem that shows that checking Accuracy-at-an-input is decidable for fixed γ. Please note that we require d to be definable in Th_+ to ensure that the distance to disagreement for a rational input is rational. All examples considered in Section 3 satisfy these constraints. In the decision procedure, the fixed value of γ, c, is encoded in the formula $\theta_{\alpha,\gamma}$. The proof can be located in Appendix F.

Theorem 15. *The problem Accuracy-at-an-input is decidable for* DiPWhile+ *programs P_ϵ and rational inputs $\bar{u}$ when*

(1) $\det(P_\epsilon), d, d'$ *are definable in* Th_+, *and*

(2) $\eta = (\theta_{\alpha,\gamma}, \psi_h, \phi_h)$ *is limit-definable and there is a rational number c such that for all $a, c', \bar{u}$, if $\theta_{\alpha,\gamma}(a, c', \bar{u})$ is true then $c' = c$.*

Program with finite outputs. We now turn our attention to programs with finite outputs. For such programs, γ is often 0. If that is the case, then we can appeal to Theorem 15 directly. However, for some examples, γ may not be 0 (for example, NoisyMax in Section 3.3).

When a program has only finite outputs, for each input $\bar{u}$, $d'(\bar{u}, \bar{v}, \bar{v}')$ can take only a finite number of distinct values. This suggests that we need to check accuracy at input u for only a finite number of possible values of γ, namely the distinct values of $d'(\bar{u}, \bar{v}, \bar{v}')$. Then as in Theorem 15, we can check for accuracy at these values of γ by setting the α parameter to be $\mathsf{dd}(P_\epsilon, u)$, the distance to disagreement for $\bar{u}$. We need a monotonicity condition that ensures the soundness of this strategy.

Definition 11. Let $\eta = (\theta_{\alpha,\gamma}, f_\beta, h_\beta)$ be a limit-definable region. Given $\bar{u}$ and non-negative $a \in \mathbb{R}^\infty$, $c \in \mathbb{R}$, let $I_{<a,c}(\bar{u})$ be the set $\{a' \mid \theta_{\alpha,\gamma}(a', c, \bar{u}) \text{ is true}, a' < a\}$. η is said to be (α,γ)-monotonic if for each $\bar{u}$ and non-negative real numbers c_1, c_2, a such that $I_{<a,c_1}(\bar{u}) \neq \emptyset$ and $I_{<a,c_2}(\bar{u}) \neq \emptyset$,

$$c_1 \leq c_2 \Rightarrow \sup(I_{<a,c_1}(\bar{u})) \leq \sup(I_{<a,c_2}(\bar{u})).$$

We have the following result whose proof can be found in Appendix G.

Theorem 16. *The problem Accuracy-at-an-input is decidable for* DiPWhile+ *programs P_ϵ and rational inputs $\bar{u}$ when (a) P_ϵ has finite outputs, (b)* $\mathsf{det}(P_\epsilon), d, d'$ *are definable in* Th_+, *and (c) η is (α,γ)-monotonic.*

When the program P_ϵ has finite inputs and finite outputs, we can invoke Theorem 16 repeatedly to check for accuracy at all possible inputs. The following is an immediate corollary of Theorem 16.

Corollary 17. *The problem Accuracy-at-all-inputs is decidable for* DiPWhile+ *programs P_ϵ when (a) P_ϵ has finite inputs and finite outputs, (b)* $\mathsf{det}(P_\epsilon), d, d'$ *are definable in* Th_+, *and (c) η is (α,γ)-monotonic.*

7 EXPERIMENTS

We implemented a simplified version of the algorithm for verifying accuracy of DiPWhile+ programs. Our tool DiPC+ handles loop-free programs with finite, discrete input domains, and whose deterministic function has discrete output. Programs with bounded loops (with constant bounds) are be handled by unrolling. The restriction that the deterministic function has discrete output does not preclude programs with real outputs, as they can be modeled in the subset of DiPWhile+ that the tool handles. We discuss this further below.

The tool takes as input a program P_ϵ parametrized by ϵ and an input-output table representing $\mathsf{det}(P_\epsilon)$ for a set of inputs, and either verifies P_ϵ to be (α,β,γ)-accurate for each given input and for all $\epsilon > 0$ or returns a counterexample, consisting of a specific input and a value for ϵ at which accuracy fails. We choose values of α,β,γ depending on the example. As accuracy claims in Section 3 show, β is typically given as a continuous function of ϵ, α and γ. For such continuous β, we can use $\alpha \leq \mathsf{dd}(P,u)$ in the definition of accuracy instead of $\alpha < \mathsf{dd}(P,u)$ (see Definition 2 on Page 6). In our experiments, we fix γ to be some integer. For a given input u, α is usually set to be the distance to disagreement for the input being checked. β can thus be viewed as a function of ϵ and the input u. The proof of Theorem 15 implies that such checks are necessary and sufficient to conclude accuracy at the given inputs for fixed γ, and all possible values of ϵ and α. In many examples, the only value for γ that needs to be verified is 0.

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 the input-output probability computations and the subsequent inequality checks. In the second phase, the generated script is run on Mathematica. We only verify accuracy-at-an-input and not accuracy-at-all-inputs as our decision procedure for the latter problem is subject to Schanuel's conjecture.

We test the ability of DiPC+ to verify accuracy-at-an-input for four examples from Section 3: Sparse, NoisyMax, Laplace Mechanism (denoted Laplace below), and NumericSparse. We also verify a variant of Sparse which we refer to as SparseVariant (the difference is discussed in 7.1 below). The pseudocode is shown in Figures 1 and 2 on Pages 8 and 10 respectively; we omit the pseudocode for Laplace and refer the reader to its description in Section 3. Three of the examples, Sparse, SparseVariant, and NoisyMax, have discrete output and thus their deterministic functions, det(Sparse), det(SparseVariant), and det(NoisyMax), are naturally modeled with a finite input-output table. The other two examples, Laplace and NumericSparse, can be modeled using a finite input-output table as follows. Given γ, we can compute the deterministic function alongside the randomized function and instrument the resulting program to check that all continuous outputs are within the error tolerance given by γ, outputting $\top$ if so. The finite input-output table can reflect this scheme by regarding the instrumented program as a computation which outputs $\top$.

Our experiments test two claims. We first test the performance of DiPC+ by measuring how running time scales with increasing input sizes and example parameters. Running times are given in Tables 2 and 3. Here the parameter m is the length of input arrays and a range $[-\ell, \ell]$ is the range of all possible integer values that can be stored in each array location. Thus, we have $(2\ell + 1)^m$ possible inputs. Hence, the tool behaves roughly polynomial in ℓ and exponential in m. In the second part of our experiments, we show that DiPC+ is able to obtain accuracy bounds that are better than those known in the literature, and generate counterexamples when accuracy claims are not true. All experiments were run on an Intel®Core i7-6700HQ @ 2.6GHz CPU with 16GB memory. In the tables, running times are reported as (T1/T2), where 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. In some tables we omit T1 when it is negligible compared to T2.

We note the following about the experimental results.

(1) DiPC+ verifies accuracy in reasonable time. The time needed to generate Mathematica scripts is significantly smaller than the time taken by Mathematica to check the scripts (i.e., T1 $\ll$ T2). Most of the time spent by Mathematica goes toward computing output probabilities.
(2) DiPC+ is able to verify that accuracy holds with smaller error probabilities than those known in the literature.
(3) Verifying accuracy is faster than verifying differential privacy. Differential privacy involves computing, for any specific input, a probability for each possible output, whereas in accuracy we only need to compute the single probability for each input-output pair given by the deterministic function.

7.1 Performance

Table 2a shows running times for Sparse and SparseVariant. SparseVariant differs from Sparse by only sampling a single noisy threshold, whereas Sparse samples a fresh noisy threshold each time it finds a query above the current noisy threshold; the pseudocode for SparseVariant is the same as Sparse (Figure 1), except for the re-sampling of r_T inside the for-loop (shown in bold). Generally, running time increases in the number of inputs that have to be verified. For both Sparse and SparseVariant, we verify $(\alpha, \beta, 0)$-accuracy for all α, β, with $\beta = 2mce^{-\alpha\epsilon/8c}$. This can be accomplished with single accuracy checks at $\alpha = \mathrm{dd}(\mathrm{Sparse}, u)$ and $\alpha = \mathrm{dd}(\mathrm{SparseVariant}, u)$, for each input u, as discussed in the proof of Theorem 15. Observe there is a substantial performance difference between Sparse and SparseVariant for $c > 1$. When analyzing SparseVariant, DiPC+ must keep track of many possible relationships between random variables and the single noisy threshold. This becomes expensive for many queries and large value of c. On the other hand, whenever Sparse samples a new noisy threshold, this has the effect of decoupling the relationship between future queries and past queries. Table 2b shows results for NoisyMax, in which we verify $(\alpha, \beta, 0)$-accuracy for all α, β with $\beta = me^{-\alpha\epsilon/2}$. Here again, a single accuracy check at $\alpha = \mathrm{dd}(\mathrm{NoisyMax}, u)$ suffices for each input u. Table 2c shows results for Laplace, where we verify $(0, \beta, \gamma)$-accuracy for all β, γ, with $\beta = ke^{-\gamma/\epsilon}$. Finally, Table 3 shows results for NumericSparse, where we verify (α, β, α)-accuracy for specific values of α, with $\beta = (2m + 1)ce^{-\alpha\epsilon/9c}$, i.e. the improved error probability bound from Section 3. In this case, the accuracy claim holds for precisely (α, β, α). This is because $\gamma = \alpha$ is no

m	1	2	2	3	3	3	4	4	4	4
c	1	1	2	1	2	3	1	2	3	4
Sparse	0s/12s	0s/45s	0s/45s	0s/97s	0s/90s	0s/89s	0s/195s	1s/189s	2s/189s	1s/195s
SparseVariant	0s/11s	0s/44s	0s/72s	0s/99s	0s/217s	0s/386s	1s/199s	1s/462s	1s/940s	1s/1467

(a) Sparse and SparseVariant

Range	$[-1, 1]$	$[-2, 2]$	$[-3, 3]$
(T1/T2)	0s/148s	1s/823s	1s/2583s

(b) NoisyMax

γ	1	2	3
$[-1, 1]$	5s	6s	6s
$[-2, 2]$	9s	9s	8s

(c) Laplace

Table 2. (a) Running times for Sparse and SparseVariant, verifying $(\alpha, \beta, 0)$-accuracy for all α, β, where $\beta = 2mce^{-\alpha\epsilon/8c}$. Accuracy is verified for all lists of m integer-valued queries ranging over $[-1, 1]$. The threshold T is set to be 0. (b) Running times for NoisyMax with $m = 3$ and varying input range, verifying $(\alpha, \beta, 0)$-accuracy for all α, β, with $\beta = me^{-\alpha\epsilon/2}$. (c) Running times for Laplace, verifying $(0, \beta, \gamma)$-accuracy with $k = 2$, $\Delta = 1$, and $\beta = ke^{-\gamma\epsilon}$.

m	1	1	1	2	2	2	3	3	3	4	4	4
α	1	1	2	1	1	2	1	1	2	1	1	2
Range	[-1,1]	[-2,2]	[-2,2]	[-1,1]	[-2,2]	[-2,2]	[-1,1]	[-2,2]	[-2,2]	[-1,1]	[-2,2]	[-2,2]
(T1/T2)	0s/18s	0s/36s	0s/19s	0s/65s	1s/287s	1s/68s	0s/178s	2s/1669s	1s/181s	2s/332s	20s/6065s	7s/350s

(a) NumericSparse with $c = 1$

m	2	2	2	3	3	3	4	4	4
α	1	1	2	1	1	2	1	1	2
Range	[-1,1]	[-2,2]	[-2,2]	[-1,1]	[-2,2]	[-2,2]	[-1,1]	[-2,2]	[-2,2]
(T1/T2)	0s/65s	0s/296s	0s/66s	0s/169s	2s/1537s	1s/170s	1s/311s	18s/5783s	3s/315s

(b) NumericSparse with $c = 2$

Table 3. Running times for NumericSparse, verifying (α, β, α)-accuracy for $\beta = (2m + 1)ce^{-\alpha\epsilon/9c}$ at specific values of α, m, c, and varying input ranges. Threshold $T = 0$ in both tables. The tables highlight scaling in run time as m and α vary. We use the improved expression for β from Corollary 7.

longer constant, and we thus cannot leverage Theorem 15 for a stronger claim. Observe that when the input range is $[-\ell, \ell]$, then $\mathrm{dd}(\mathrm{NumericSparse}, x)$ can take any integer value between 0 and ℓ. Thus, even though we have set α to be the same value in each individual experiment (and not the distance to disagreement for the input), we vary α across the experiments so as to ensure that for each input x, accuracy is eventually verified when α is set to $\mathrm{dd}(\mathrm{NumericSparse}, u)$.

7.2 Improved Accuracy Bounds and Counterexamples

In Tables 4a, 4b, and 4c, we summarize the results of verifying better accuracy bounds than those known in the literature for Sparse, SparseVariant, NoisyMax, and NumericSparse. For each algorithm and parameter choice, we display the best accuracy bound obtained by searching for the largest integer κ for which DiPC+ was able to verify $(\alpha, \beta/\kappa, \gamma)$-accuracy, where α, β, γ are the corresponding values from the experiments in Subsection 7.1. In each row, we include a counterexample returned by the tool after a failed attempt to verify $(\alpha, \beta/\kappa{+}1, \gamma)$-accuracy. Each counterexample consists of both an ϵ and a specific input u at which the accuracy check failed.

8 RELATED WORK

Accuracy proofs. The Union Bound logic of Barthe et al. [2016b] is a program logic for upper bounding errors in probabilistic computation. The logic is a form of lightweight probabilistic

Algorithm	c	β	Best	(T1/T2)	Counterexample	(T1/T2)
Sparse	1	$6e^{-\alpha\epsilon/8}$	$1/6$	0s/101s	$1/7$, $u = [-1, -1, 1]$, $\epsilon = 17/10$	0s/110s
Sparse	2	$12e^{-\alpha\epsilon/16}$	$1/12$	0s/91s	$1/13$, $u = [1, -1, 1]$, $\epsilon = 50/19$	0s/95s
SparseVariant	1	$6e^{-\alpha\epsilon/8}$	$1/6$	0s/99s	$1/7$, $u = [-1, -1, 0]$, $\epsilon = 67/106$	0s/111s
SparseVariant	2	$12e^{-\alpha\epsilon/16}$	$1/13$	0s/220s	$1/14$, $u = [-1, -1, -1]$, $\epsilon = 17/13$	0s/216s

(a) Improved accuracy bounds for Sparse and SparseVariant, with $m = 3$.

m	β	Best	(T1/T2)	Counterexample	(T1/T2)
3	$3e^{-\alpha\epsilon/2}$	$1/4$	0s/154s	$1/5$, $u = [-1, 0, 0]$, $\epsilon = 27/82$	0s/176s
4	$4e^{-\alpha\epsilon/2}$	$1/4$	0s/874s	$1/5$, $u = [0, 0, 1, 0]$, $\epsilon = 52/23$	0s/910s

(b) Improved accuracy bounds for NoisyMax.

α	c	β	Best	(T1/T2)	Counterexample	(T1/T2)
1	1	$7e^{-\epsilon/9}$	$1/3$	1s/174s	$1/4$, $u = [-1, -1, 1]$, $\epsilon = 37$	1s/173s
1	2	$14e^{-\epsilon/18}$	$1/5$	1s/162s	$1/6$, $u = [-1, 1, 1]$, $\epsilon = 59$	1s/167s

(c) Improved accuracy bounds for NumericSparse, with $m = 3$.

Table 4. Accuracy is checked for all vectors of length m with entries ranging over [-1,1]. Best column displays $1/\kappa$, where κ is the largest integer for which the tool verified accuracy. The counterexample column displays $1/\kappa{+}1$ along with ϵ and input u at which the accuracy check failed. We include β for clarity because of its dependence on c.

program logic: assertions are predicates on states, and probabilities are only tracked through an index that accounts for the cumulative error. The Union bound logic has been used to prove accuracy bounds for many of the algorithms considered in this paper. However, the proofs must be constructed manually, often at considerable cost. Moreover, all reasoning about errors use union bounds, so precise bounds that use concentration inequalities are out of scope of the logic. Finally, the proof system is sound, but incomplete. For instance, the proof system cannot deal with arbitrary loops. In contrast, our language allows for arbitrary loops and we provide a decision procedure for accuracy. Trace Abstraction Modulo Probability (TAMP) in Smith et al. [2019], is an automated proof technique for accuracy of probabilistic programs. TAMP generalizes the trace abstraction technique of Heizmann et al. [2009] to the probabilistic setting. TAMP follows the same lightweight strategy as the union bound logic, and uses failure automata to separate between logical and probabilistic reasoning. TAMP has been used for proving accuracy of many algorithms considered in this paper. However, TAMP suffers from similar limitations as the Union Bound logic (except of course automation): it is sound but incomplete, and cannot deal with arbitrary loops and concentration inequalities.

Privacy proofs. There is a lot of work on verification and testing of privacy guarantees [Albarghouthi and Hsu 2018; Barthe et al. 2020a, 2013; Bichsel et al. 2018; Ding et al. 2018; Gaboardi et al. 2013; Reed and Pierce 2010; Zhang and Kifer 2017]. We refer to [Barthe et al. 2016c] for an overview.

Program analysis. There is a large body of work that lifts to the probabilistic setting classic program analysis and program verification techniques, including deductive verification [Kaminski 2019; Kozen 1985; Morgan et al. 1996], model-checking [Katoen 2016; Kwiatkowska et al. 2010], abstract interpretation [Cousot and Monerau 2012; Monniaux 2000], and static program analysis [Sankaranarayanan et al. 2013; Wang et al. 2018]. Some of these approaches rely on advanced techniques from probability theory, including concentration inequalities [Sankaranarayanan 2020] and martingales [Barthe et al. 2016a; Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016; Kura et al. 2019; Wang et al. 2020] for better precision.

These techniques can compute sound upper bounds of the error probability for a general class of probabilistic programs. However, this does not suffice to make them immediately applicable to our setting, since our definition of accuracy involves the notion of distance to disagreement. Moreover, these works generally do not address the specific challenge of reasoning in the theory of reals with exponentials. Finally, these techniques cannot be used to prove the violation of accuracy claims; our approach can both prove and siprove accuracy claims.

Hyperproperties. Hyperproperties [Clarkson and Schneider 2010] are a generalization of program properties and encompass many properties of interest, particularly in the realm of security and privacy. Our definition of accuracy falls in the class of 3-properties, as it uses two executions of $det(P)$ for defining distance to disagreement, and a third execution of P for quantifying the error.

There is a large body of work on verifying hyperproperties. While the bulk of this literature is in a deterministic setting, there is a growing number of logics and model-checking algorithms from probabilistic hyperproperties [Ábrahám and Bonakdarpour 2018; Dimitrova et al. 2020; Wang et al. 2019]. To our best knowledge, these algorithms do not perform parametrized verification, and cannot prove accuracy for all possible values of ϵ.

9 CONCLUSIONS

We have introduced a new uniform definition of accuracy, called (α, β, γ)-accuracy, for differential privacy algorithms. This definition adds an additional parameter α, that accounts for distance to disagreement, to the traditional parameters β and γ. This uniform, generalized definition can be used to unify under a common scheme different accuracy definitions used in the literature that quantify the probability of getting approximately correct answer, including *ad hoc* definitions for classical algorithms such as AboveThreshold, Sparse, NumericSparse, NoisyMax and others. Using the (α, β, γ) frame work of accuracy we were able to improve the accuracy results for NumericSparse. We have shown that checking (α, β, γ)-accuracy is decidable for a non-trivial class of algorithms with finite number of real inputs and outputs, that are parametrized by privacy parameter ϵ, described in our expanded programming language DiPWhile+, for all values of ϵ within a given interval I, assuming that Schanuel's conjecture. We have also shown that the problem of checking accuracy at a single input decidable under reasonable assumptions without assuming Schanuel's conjecture for programs in DiPWhile+, even when the inputs and outputs can take any real values. This implies that checking accuracy is decidable for programs whose inputs and outputs take values in finite domains. Finally, we presented experimental results implementing our approach by adapting DiPC to check accuracy at specified inputs for AboveThreshold, Sparse, NumericSparse and NoisyMax.

In the future, it would be interesting to study how our decision procedures could be used for automatically proving concentration bounds, and how it could be integrated in existing frameworks for accuracy of general-purpose probabilistic computations. It would also be interesting to extend our results and the results from Barthe et al. [2020a] to accommodate unbounded number of inputs and outputs, and other probability distributions, e.g. Gaussian mechanism. On a more practical side, it would be interesting to study the applicability of our techniques in the context of the accuracy first approach from Ligett et al. [2017].

ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for their interesting and useful comments. This work was partially supported by National Science Foundation grants NSF CNS 1553548, NSF CCF 1900924, NSF CCF 1901069 and NSF CCF 2007428.

REFERENCES

Erika Ábrahám and Borzoo Bonakdarpour. 2018. HyperPCTL: A Temporal Logic for Probabilistic Hyperproperties. In *Quantitative Evaluation of Systems - 15th International Conference, QEST 2018, Beijing, China, September 4-7, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11024)*, Annabelle McIver and András Horváth (Eds.). Springer, 20–35. https://doi.org/10.1007/978-3-319-99154-2_2

Aws Albarghouthi and Justin Hsu. 2018. Synthesizing coupling proofs of differential privacy. *PACMPL* 2, POPL (2018), 58:1–58:30. https://doi.org/10.1145/3158146

Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2020a. Deciding Differential Privacy for Programs with Finite Inputs and Outputs. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller (Eds.). ACM, 141–154. https://doi.org/10.1145/3373718.3394796

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

Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. 2016a. Synthesizing Probabilistic Invariants via Doob's Decomposition. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9779)*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer, 43–61. https://doi.org/10.1007/978-3-319-41528-4_3

Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016b. A program logic for union bounds. In *International Colloquium on Automata, Languages and Programming (ICALP), Rome, Italy.* arXiv:Yes http://arxiv.org/abs/1602.05681

Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin C. Pierce. 2016c. Programming language techniques for differential privacy. *SIGLOG News* 3, 1 (2016), 34–53. https://dl.acm.org/citation.cfm?id=2893591

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. http://software.imdea.org/~bkoepf/papers/toplas13.pdf

Raghav Bhaskar, Srivatsan Laxman, Adam D. Smith, and Abhradeep Thakurta. 2010. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang (Eds.). ACM, 503–512. https://doi.org/10.1145/1835804.1835869

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, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 508–524. https://doi.org/10.1145/3243734.3243863

Avrim Blum, Katrina Ligett, and Aaron Roth. 2013. A learning theory approach to noninteractive database privacy. *J. ACM* 60, 2 (2013), 12:1–12:25. https://doi.org/10.1145/2450142.2450148

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8044)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

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. http://eprint.iacr.org/2010/076.pdf

Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 327–342. https://doi.org/10.1145/2837614.2837639

Michael R. Clarkson and Fred B. Schneider. 2010. Hyperproperties. *J. Comput. Secur.* 18, 6 (2010), 1157–1210. https://doi.org/10.3233/JCS-2009-0393

Patrick Cousot and Michael Monerau. 2012. Probabilistic Abstract Interpretation. In *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7211)*, Helmut Seidl (Ed.). Springer, 169–193. https://doi.org/10.1007/978-3-642-28869-2_9

Rayna Dimitrova, Bernd Finkbeiner, and Hazem Torfah. 2020. Probabilistic Hyperproperties of Markov Decision Processes. In *Automated Technology for Verification and Analysis*, Dang Van Hung and Oleg Sokolsky (Eds.). Springer International Publishing, Cham, 484–500.

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. https://doi.org/10.1145/3243734.3243818

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. http://dx.doi.org/10.1007/11681878_14

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. http://dx.doi.org/10.1561/0400000042

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. http://dl.acm.org/citation.cfm?id=2429113

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. http://arxiv.org/pdf/0903.4510v2

Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2009. Refinement of Trace Abstraction. In *Static Analysis, 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5673)*, Jens Palsberg and Zhendong Su (Eds.). Springer, 69–85. https://doi.org/10.1007/978-3-642-03237-0_7

Benjamin Lucien Kaminski. 2019. *Advanced weakest precondition calculi for probabilistic programs*. Ph.D. Dissertation. RWTH Aachen University, Germany. http://publications.rwth-aachen.de/record/755408

Joost-Pieter Katoen. 2016. The Probabilistic Model Checking Landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 31–45. https://doi.org/10.1145/2933575.2934574

Dexter Kozen. 1985. A Probabilistic PDL. *J. Comput. System Sci.* 30, 2 (1985), 162–178.

Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11428)*, Tomás Vojnar and Lijun Zhang (Eds.). Springer, 135–153. https://doi.org/10.1007/978-3-030-17465-1_8

Marta Kwiatkowska, Gethin Norman, and David Parker. 2010. Advances and challenges of probabilistic model checking. In *48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 1691–1698.

Serge Lang. 1966. *Introduction to Transcendental Numbers*. Addison–Wesley.

Katrina Ligett, Seth Neel, Aaron Roth, Bo Waggoner, and Steven Z. Wu. 2017. Accuracy First: Selecting a Differential Privacy Level for Accuracy Constrained ERM. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 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.). 2566–2576.

Angus MacIntyre and Alex J. Wilkie. 1996. On the decidability of the real exponential field. In *Kreiseliana. About and Around Georg Kreisel*, Piergiorgio Odifreddi (Ed.). A.K. Peters, 441–467.

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

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. https://doi.org/10.1109/FOCS.2007.41

David Monniaux. 2000. Abstract Interpretation of Probabilistic Semantics. In *Static Analysis, 7th International Symposium, SAS 2000, Santa Barbara, CA, USA, June 29 - July 1, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1824)*, Jens Palsberg (Ed.). Springer, 322–339. https://doi.org/10.1007/978-3-540-45099-3_17

Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic Predicate Transformers. *ACM Transactions on Programming Languages and Systems* 18, 3 (1996), 325–353.

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.

Jason Reed and Benjamin C. Pierce. 2010. Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming* (Baltimore, Maryland, USA) *(ICFP '10)*. Association for Computing Machinery, New York, NY, USA, 157–168. https://doi.org/10.1145/1863543.1863568

Sriram Sankaranarayanan. 2020. Quantitative Analysis of Programs with Probabilities and Concentration of Measure Inequalities. In *Foundations of Probabilistic Programming*, Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva (Eds.). Cambridge University Press, TBA.

Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, Hans-Juergen Boehm and Cormac Flanagan (Eds.). ACM, 447–458. https://doi.org/10.1145/2491956.2462179

Calvin Smith, Justin Hsu, and Aws Albarghouthi. 2019. Trace abstraction modulo probability. *PACMPL* 3, POPL (2019), 39:1–39:31. https://dl.acm.org/citation.cfm?id=3290352

A. Tarski. 1951. *A decision method for Elementary Algebra and Geometry*. University of California Press.

Elisabet Lobo Vesga, Alejandro Russo, and Marco Gaboardi. 2019. A Programming Framework for Differential Privacy with Accuracy Concentration Bounds. *CoRR* abs/1909.07918 (2019). arXiv:1909.07918 http://arxiv.org/abs/1909.07918

Di Wang, Jan Hoffmann, and Thomas W. Reps. 2018. PMAF: an algebraic framework for static analysis of 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, 513–528. https://doi.org/10.1145/3192366.3192408

Di Wang, Jan Hoffmann, and Thomas W. Reps. 2020. Tail Bound Analysis for Probabilistic Programs via Central Moments. *CoRR* abs/2001.10150 (2020). arXiv:2001.10150 https://arxiv.org/abs/2001.10150

Yu Wang, Siddhartha Nalluri, Borzoo Bonakdarpour, and Miroslav Pajic. 2019. Statistical Model Checking for Probabilistic Hyperproperties. *CoRR* abs/1902.04111 (2019). arXiv:1902.04111 http://arxiv.org/abs/1902.04111

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, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 888–901. http://dl.acm.org/citation.cfm?id=3009884