Deciding Differential Privacy of Online Algorithms with Multiple Variables

Rohit Chadha
chadhar@missouri.edu
University of Missouri
Columbia, Missouri, USA

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

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

Bishnu Bhusal
bhusalb@mail.missouri.edu
University of Missouri
Columbia, Missouri, USA

ABSTRACT

We consider the problem of checking the differential privacy of *online* randomized algorithms that process a stream of inputs and produce outputs corresponding to each input. This paper generalizes an automaton model called DiP automata [10] to describe such algorithms by allowing multiple real-valued storage variables. A DiP automaton is a parametric automaton whose behavior depends on the privacy budget ϵ. An automaton $\mathcal{A}$ will be said to be differentially private if, for some $\mathfrak{D}$, the automaton is $\mathfrak{D}\epsilon$-differentially private for all values of $\epsilon > 0$. We identify a precise characterization of the class of all differentially private DiP automata. We show that the problem of determining if a given DiP automaton belongs to this class is PSPACE-complete. Our PSPACE algorithm also computes a value for $\mathfrak{D}$ when the given automaton is differentially private. The algorithm has been implemented, and experiments demonstrating its effectiveness are presented.

CCS CONCEPTS

• **Security and privacy → Logic and verification**; **Formal security models**.

KEYWORDS

Differential Privacy, Verification, Automata, Decision procedure

ACM Reference Format:
Rohit Chadha, A. Prasad Sistla, Mahesh Viswanathan, and Bishnu Bhusal. 2023. Deciding Differential Privacy of Online Algorithms with Multiple Variables. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623170

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623170

1 INTRODUCTION

Differential privacy [18, 20] is a popular requirement that is demanded of algortihms that analyze data containing sensitive personal information of individuals. A data analysis that meets the high bar of differential privacy guarantees the privacy of individuals. However, ensuring differential privacy is difficult, subtle and error-prone — relatively minor tweaks to correct algorithms can lead to the loss of privacy as demonstrated by the examples in [19, 26]. Though the problem of checking the differential privacy of a program is in general undecidable [2], the importance of the problem has led to extensive investigation in the last 15 years; see Section 8 for a short overview of work in this space.

In this paper, we look at the problem of verifying the differential privacy of online algorithms. An online algorithm is one that processes an unbounded (but finite) stream of inputs, samples from distributions, and produces outputs in response to the inputs. The stream of inputs is a sequence of real numbers that are answers to queries to a database. A novel approach using automata to describe and study such algorithms was proposed in [10]. It was shown that checking differential privacy of algorithms described by such automaton is in linear time. Remarkably the verification procedure in [10] checks some properties of the underlying graph of the automaton and does not explicitly reason about probabilities. However, the automaton model in [10] has one serious limitation — only one storage variable is available, and hence only one previously sampled value can be remembered.

Contributions. We extend the line of research initiated in [10] by generalizing the automata model in [10] to allow for multiple real-valued storage variables. A DiP automaton (DiPA for short) [1] is a parametric automaton (depending on privacy budget ϵ) with finitely many control states that process an unbounded (but finite) stream of real values that represent answers to queries asked of a database. A DiPA can sample real values from Laplace distributions whose mean may depend on the value read, and DiPA has finitely many real-valued variables in which they can store values they sample in each step (which in turn depend on the input read). Transitions depend on the current control state, the values stored, and the input read, which influences the values sampled. In response to an input,

[1] Even though the automata model in this paper has the same name as the one in [10], the generalization significantly extends the expressive power of the model.

they produce an output that is either a symbol from a finite set or a real number.

We show that, even in the case of automata with multiple storage variables, the problem of determining whether a given DiPA $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private for some constant $\mathfrak{D} > 0$ (independent of ϵ) and all $\epsilon > 0$, can be reduced to checking graph-theoretic conditions. These conditions demand the absence of certain paths, cycles, and interactions among them. However, unlike the single variable automata case [10], these paths and cycles cannot be captured only by considering the underlying graph of the automata. Instead, we use an auxiliary graph to capture these undesirable paths and cycles precisely. This is a non-trivial extension of [10]; for a more detailed comparison with [10], see Section 8. An automaton $\mathcal{A}$ is said to be *well-formed* if it does not have any of these undesirable paths or cycles. We show that a well-formed DiPA is differentially private; thus, well-formedness is a sufficient condition to guarantee privacy. Conversely, we show that if additionally, for every state of $\mathcal{A}$, the transitions of $\mathcal{A}$ from that state have distinct outputs (called *output distinct*), then well-formedness is also necessary to guarantee differential privacy. In other words, a DiPA $\mathcal{A}$, having distinct outputs on transitions from any state, that is differentially private is well-formed. These proofs of necessity and sufficiency require novel ideas that are a significant extension of the techniques presented in [10]; once again see Section 8 for more details.

Next, we show that there is a PSPACE algorithm that checks if a DiPA $\mathcal{A}$ is well-formed. This algorithm additionally computes a value for $\mathfrak{D}$ that shows that $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private for all ϵ. We also show that checking differential privacy of output-distinct DiPA is PSPACE-hard, thus establishing the optimality of our verification algorithm.

We have implemented our algorithm in a tool called DiPAut. Our experiments show that the approach scales and that our algorithm produces known estimates for $\mathfrak{D}$. It successfully proves differential privacy and identifies violations of privacy in various examples. The tool is evaluated for scalability with respect to both the number of states and variables. Despite the PSPACE-hardness, the tool is able to perform well in our experiments. We compare DiPAut with CheckDP [29], a state-of-the-art tool to check differential privacy. DiPAut significantly outperforms CheckDP in all our experiments. The tool DiPAut is available to download at [7].

Organization. The rest of the paper is organized as follows. Section 2 introduces basic notation and definitions used in the paper. Our model of DiP automaton extended with multiple variables is introduced in Section 3. Section 4 defines well-formed DiPA, which is a (almost) precise characterization of differentially private automata. We show that well-formed automata are differentially private in Section 5; and show that checking well-formedness is PSPACE-complete. Section 6 shows that differentially private automata that have distinct outputs on transitions are well-formed. PSPACE-hardness of checking differential privacy is also presented in this section. Experimental results are presented in Section 7. Closely related work is discussed in Section 8. We discuss on the restrictions placed on the automata and the adjacency relations used in the paper. Finally we present our conclusions (Section 10).

For lack of space reasons, some proofs are omitted and can be found in [11].

2 PRELIMINARIES

The definitions and notations in this section are borrowed from [10]. Let $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{Q}^{\geq 0}, \mathbb{R}, \mathbb{R}^{>0}$ denote the set of natural numbers, integers, rational numbers, non-negative rationals, real numbers, and positive real numbers, respectively. In addition, $\mathbb{R}_\infty$ will denote the set $\mathbb{R} \cup \{-\infty, \infty\}$, where $-\infty$ is the smallest and ∞ is the largest element in $\mathbb{R}_\infty$. For a real number $x \in \mathbb{R}$, $|x|$ denotes its absolute value.

Sequences. For a set Σ, Σ^* denotes the set of all finite sequences/strings over Σ. We use λ to denote the empty sequence/string over Σ. For two sequences/strings $\rho, \sigma \in \Sigma^*$, we use their juxtaposition $\rho\sigma$ to indicate the sequence/string obtained by concatenating them in order. Consider $\sigma = a_0 a_1 \cdots a_{n-1} \in \Sigma^*$ (where $a_i \in \Sigma$). We use $|\sigma|$ to denote its length n and use $\sigma[i]$ to denote its ith symbol a_i. The substring $a_i a_{i+1} \cdots a_{j-1}$ from position i (inclusive) to j (not inclusive) will be denoted as $\sigma[i:j]$; if $j \leq i$ then $\sigma[i:j] = \lambda$. Thus, $\sigma[0:|\sigma|] = \sigma$. The suffix starting at position j will be denoted as $\sigma[j:]$, i.e., $\sigma[j:] = \sigma[j:|\sigma|]$. For any partial function $f : A \hookrightarrow B$, where A, B are some sets, we let $\mathrm{dom}(f)$ be the set of $x \in A$ such that $f(x)$ is defined.

Laplace Distribution. Differential privacy mechanisms often add noise by sampling values from the *Laplace distribution*. The distribution, denoted $\mathsf{Lap}(k, \mu)$, is parameterized by two values: $k \geq 0$ which is called the scaling parameter, and μ which is the mean. The probability density function of $\mathsf{Lap}(k, \mu)$, denoted $f_{k,\mu}$, is given by $f_{k,\mu}(x) = \frac{k}{2} e^{-k|x-\mu|}$, where e is the Euler constant.

Differential Privacy. Differential privacy [18] is a framework that enables statistical analysis of databases containing sensitive, personal information of individuals while ensuring that the privacy of individuals is not adversely affected by the results of the analysis. In the differential privacy framework, a randomized algorithm, M, called the *differential privacy mechanism*, mediates the interaction between a (possibly dishonest) data analyst asking queries and a database D responding with answers. Queries are deterministic functions and typically include aggregate questions about the data, like the mean etc. In response to such a sequence of queries, M responds with a series of answers computed using the actual answers from the database and random sampling, resulting in "noisy" answers. Thus, M provides privacy at the cost of accuracy. Typically, M's noisy response depends on a *privacy budget* $\epsilon > 0$.

Differential privacy captures the privacy guarantees for individuals whose information is in the database D. For an individual i, let $D \setminus \{i\}$ denote the database where i's information has been removed. A secure mechanism M ensures that for any individual i in D, and any sequence of possible outputs $\overline{o}$, the probability that M outputs $\overline{o}$ on a sequence of queries is approximately the same whether the interaction is with the database D or with $D \setminus \{i\}$. To capture this definition formally, we need to characterize the inputs on which M is required to behave similarly. Inputs to a differential privacy mechanism can be seen as answers from the database to a sequence of queries asked by the data analyst. If queries are aggregate queries, then answers to q on D and $D \setminus \{i\}$ (for individual i) are likely to

be away by at most 1. [2] This intuition leads to the following often-used definition of *adjacency* that characterizes inputs on which the differential privacy mechanism M is expected to behave similarly; for example this definition is used in SVT [1, 17, 19, 20, 26] and NumericSparse [20].[3] We assume that at each step, the differential privacy mechanism either gets a real number as input (answer to an aggregate query) or is asked to respond without an answer from the database which is encoded as τ.

Definition 1. Sequences $\rho, \sigma \in (\mathbb{R} \cup \{\tau\})^*$ are *adjacent* if $|\rho| = |\sigma|$ and for each $i \leq |\rho|$ (a) $\rho[i] \in \mathbb{R}$ iff $\sigma[i] \in \mathbb{R}$ and (b) if $\rho[i] \in \mathbb{R}$ then $|\rho[i] - \sigma[i]| \leq 1$.

We are now ready to formally define the notion of privacy which uses Definition 1. In response to a sequence of inputs, a differential privacy mechanism produces a sequence of outputs from the set (say) Γ. Since a differential privacy mechanism M is a randomized algorithm, it will induce a probability distribution on Γ^*.

Definition 2 (ϵ-differential privacy)**.** A randomized algorithm M with input in $(\mathbb{R} \cup \{\tau\})^*$ and output in Γ^* is said to be *ϵ-differentially private* if for all measurable sets $S \subseteq \Gamma^*$ and adjacent $\rho, \sigma \in \mathbb{R}^*$ (Definition 1),

$$\text{Prob}[M(\rho) \in S] \leq e^{\epsilon} \text{ Prob}[M(\sigma) \in S].$$

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

low $\leftarrow$ Lap($\frac{\epsilon}{4}, T_\ell$)
high $\leftarrow$ Lap($\frac{\epsilon}{4}, T_u$)
for $i \leftarrow 1$ **to** N **do**
  r $\leftarrow$ Lap($\frac{\epsilon}{4}, q[i]$)
  if (r $\geq$ low) $\wedge$ (r $<$ high) **then**
    $out[i] \leftarrow \perp$
  else if (r $\geq$ low) $\wedge$ (r $\geq$ high) **then**
    $out[i] \leftarrow \top_1$
    exit
  else if (r $<$ low) $\wedge$ (r $<$ high) **then**
    $out[i] \leftarrow \top_2$
    exit
  end
end

Algorithm 1: Range query algorithm

Example 1. Consider the following problem. Given a sequence of answers to queries (array $q[1:N]$) and an interval $[T_\ell, T_u)$ given by thresholds T_ℓ and T_u, determine the first time a query answer lies outside this interval; indicate (through the output) whether the query answer is $\geq T_u$ or $\leq T_\ell$ at this point. A differentially private algorithm to solve this problem is shown as Algorithm 1. The algorithm starts by adding noise to both T_ℓ and T_u to get a perturbed interval defined by numbers low and high. After that the algorithm perturbs each query answer and stores the result in r, and then checks if r lies between low and high. If it does, the algorithm outputs $\perp$ and processes the next query answer. Otherwise, if r is larger than both low and high it outputs $\top_1$ and stops. On the other hand, if r is less than both low and high then it outputs $\top_2$ and halts. The algorithm's behavior depends on the value of ϵ. It can be shown that for each value of ϵ, the algorithm for that value of ϵ is ϵ-differentially private.

[2] The difference in general can be bounded by a constant Δ.

[3] Please see the discussion of SVT on pages 56 and 57 of [20] and its description on pages 58, 62, and 64. For simplicity, it is assumed that these queries are 1-sensitive. So, by considering SVT as an algorithm that works directly on the sequence of the outputs of queries, we get naturally the adjacency relation used here.

3 DIPA

DiP (**Di**fferentially **P**rivate) automata (DiPAs for short) are an automata-based model introduced in [10] to describe some differential privacy mechanisms. They process an input string $\sigma \in (\mathbb{R} \cup \{\tau\})^*$ by sampling values from the Laplace distribution, using real variables to store information during the computation, and producing a sequence of outputs. The model introduced in [10] had only *one* storage variable. In this paper, we generalize this model naturally to allow *multiple* real-valued storage variables. However, as discussed in Section 8, both the characterization of differentially private algorithms described by them and the proofs of decidability are a non-trivial extension of the single variable model.

3.1 Syntax

A DiP automaton is a *parametric* automaton whose behavior depends on a parameter ϵ (the privacy budget). It has finitely many control states and finitely many real-valued variables $\mathsf{x}_1, \mathsf{x}_2, \ldots \mathsf{x}_k$ that are used to store information during the computation. At each step, the automaton freshly samples two real values from Laplace distributions whose parameters depend on ϵ, and these sampled values are stored in the (additional) variables insample and insample$'$. Given an input $\sigma \in (\mathbb{R} \cup \{\tau\})^*$, a DiPA does the following in each step.

(1) Two values are drawn from the distributions Lap($d\epsilon, \mu$) and Lap($d'\epsilon, \mu'$) and stored in the variables insample and insample$'$, respectively. The scaling factors d, d' and means μ, μ' of these distributions depend on the current state.
(2) The states of the automaton are partitioned into *input* states and *non-input* states. At a non-input state, the automaton expects to read τ from the input. On the other hand, at an input state, it expects to read a real number, say a, and it updates insample and insample$'$ by adding a to them. The properties of the Laplacian distribution imply that the distribution of insample + a (insample$'$ + a) is the same as the distribution of Lap($d\epsilon, \mu + a$) (Lap($d\epsilon, \mu' + a$) respectively).
(3) A transition changes the control state and outputs a value. The value output could either be a symbol from a finite set or one of the two real numbers insample and insample$'$ that are sampled in this step. At an input state, the transition is guarded by a Boolean condition that depends on the result of comparing the sampled value insample with the stored values x_i ($1 \leq i \leq k$). It is possible that for certain values of x_i ($1 \leq i \leq k$) and insample, no transition is enabled from the current state. In such a case, the computation ends.
(4) Finally, the automaton may choose to store the sampled value insample in any of the variables x_i ($1 \leq i \leq k$).

We now formally define DiP automaton capturing the above intuition. First, some necessary notation. Let $\mathcal{G}'$ be the set of constraints defined as $\mathcal{G}' = \{\mathsf{insample} \geq \mathsf{x}_i \mid 1 \leq i \leq k\} \cup \{\mathsf{insample} < \mathsf{x}_i \mid 1 \leq i \leq k\}$. Let $\mathcal{G}''$ be the set of conditions formed by taking conjunctions of two or more constraints in $\mathcal{G}'$ such that both $\mathsf{insample} \geq \mathsf{x}_i$ and $\mathsf{insample} < \mathsf{x}_i$ don't appear for any $1 \leq i \leq k$. Finally, let $\mathcal{G} = \{\mathsf{true}\} \cup \mathcal{G}' \cup \mathcal{G}''$; these are the constraints that *guard* transitions in a DiPA.[4]

Definition 3 (DiPA). A *DiP automaton* $\mathcal{A} = (Q, \Gamma, q_{\mathsf{init}}, X, P, \delta)$ where

- Q is a finite set of states partitioned into two sets: the set of input states Q_{in} and the set of non-input states Q_{non},
- Γ is a finite output alphabet,
- $q_{\mathsf{init}} \in Q$ is the initial state,
- $X = \{\mathsf{insample}, \mathsf{insample}'\} \cup \{\mathsf{x}_i \mid 1 \leq i \leq k\}$ is the set of variables; we will use $\mathsf{stor} = \{\mathsf{x}_i \mid 1 \leq i \leq k\}$ to denote the *storage* variables,
- $P : Q \to \mathbb{Q}^{\geq 0} \times \mathbb{Q} \times \mathbb{Q}^{\geq 0} \times \mathbb{Q}$ is the parameter function that assigns to each state a 4-tuple (d, μ, d', μ'), where insample is sampled from $\mathsf{Lap}(d\epsilon, \mu)$ and insample′ is sampled from $\mathsf{Lap}(d'\epsilon, \mu')$,
- and $\delta : (Q \times \mathcal{G}) \hookrightarrow (Q \times (\Gamma \cup \{\mathsf{insample}, \mathsf{insample}'\}) \times \{\mathsf{true}, \mathsf{false}\}^k)$ is the transition (partial) function that given a current state and the result of comparing each x_i $(1 \leq i \leq k)$ with insample, determines the next state, the output, and whether the variables x_i should be updated to store insample. The output could either be a symbol from Γ or the values insample and insample′ that were sampled.

In addition, the transition function δ satisfies the following two conditions.

Determinism: For any state $q \in Q$, if $\delta(q, c)$ and $\delta(q, c')$ are defined for $c, c' \in \mathcal{G}$ then either $c = c'$ or $c \wedge c'$ is unsatisfiable. That is, from any state, at most one transition is enabled at any time.

Non-input transitions: From any $q \in Q_{\mathsf{non}}$, if $\delta(q, c)$ is defined, then $c = \mathsf{true}$; that is, there is at most one transition from a non-input state which is always enabled.

Remark. Although insample′ is never used in comparisons, it is nevertheless needed to model examples such as Num-Sparse (See [20]). insample′ is often used in algorithms when we want to output the noisy input value in a differentially private fashion. Outputting insample instead of insample′ can violate differential privacy, as insample may have been used in other comparisons: See the definition of privacy violating path (Definition 11 in Section 4); also [26].

Before concluding this section, it is useful to introduce some notation and terminology for transitions. A quintuple $t = (q, c, q', o, b)$ denotes a transition of $\mathcal{A}$ if $\delta(q, c) = (q', o, b)$, where $b = (b_1, b_2, \ldots b_k) \in \{\mathsf{true}, \mathsf{false}\}^k$. For such a transition, $\mathsf{src}(t) = q$ denotes the *source*, $\mathsf{trg}(t) = q'$ the *target*, $\mathsf{out}(t) = o \in \Gamma \cup \{\mathsf{insample}, \mathsf{insample}'\}$ the *output*, and $\mathsf{guard}(t) = c$ the *guard*. Based on the guard c and the Booleans b, we can associate the following sets of variables with transition t.

$$\begin{aligned}
\mathsf{smallv}(t) &= \{\mathsf{x} \in \mathsf{stor} \mid \mathsf{insample} \geq \mathsf{x} \text{ is a conjunct of } c\} \\
\mathsf{largev}(t) &= \{\mathsf{x} \in \mathsf{stor} \mid \mathsf{insample} < \mathsf{x} \text{ is a conjunct of } c\} \\
\mathsf{usedv}(t) &= \mathsf{smallv}(t) \cup \mathsf{largev}(t) \\
\mathsf{assignv}(t) &= \{\mathsf{x}_i \mid b_i = \mathsf{true}\} \\
\mathsf{nonassignv}(t) &= \{\mathsf{x}_i \mid b_i = \mathsf{false}\}
\end{aligned}$$

Intuitively, $\mathsf{smallv}(t)$ ($\mathsf{largev}(t)$) are the storage variables that lower bound (upper bound) insample if the guard is satisfied; $\mathsf{usedv}(t)$ are the storage variables that are referenced in the guard of t; $\mathsf{assignv}(t)$ are the variables that are set by t; and $\mathsf{nonassignv}(t)$ are the variables that are left unchanged by t. For any i, if $\mathsf{x}_i \in \mathsf{assignv}(t)$ then t sets $\mathsf{x}_i = \mathsf{insample}$ during the transition and hence t is an *assignment transition* for variable x_i. Finally, if $\mathsf{src}(t) = q \in Q_{\mathsf{in}}$ then t is said to be *input transition* and if $q \in Q_{\mathsf{non}}$ then t is a *non-input transition*.

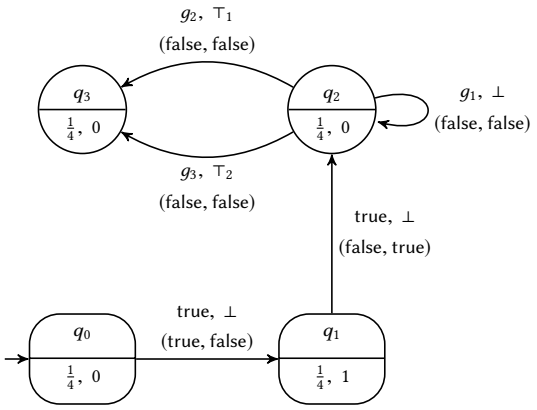

Figure 1: **DiPA $\mathcal{A}_{\mathsf{range}}$ modeling Algorithm 1. Threshold T_ℓ is set to 0 (sampling mean of** insample **in q_0) and T_u is set to 1 (sampling mean of** insample **in q_1). The guards g_1 = (insample $\geq \mathsf{x}_1$) $\wedge$ (insample $< \mathsf{x}_2$), g_2 = (insample $\geq \mathsf{x}_1$) $\wedge$ (insample $\geq \mathsf{x}_2$), and g_3 = (insample $< \mathsf{x}_1$) $\wedge$ (insample $< \mathsf{x}_2$).**

Example 2. The differential privacy mechanism in Example 1 can be modeled as a DiPA. This is shown in Figure 1. We will use these conventions when drawing DiPAs in this paper. Input states will be represented as circles, while non-input states will be drawn as rectangles. The name of each state is written above the line, while the scaling factor d and mean μ of the distribution used to sample insample is written below the line. The parameters d' and μ' for sampling insample′ are not shown in the figures, but will be mentioned in the caption and text when they are important; they are relevant only when insample′ is output on a transition. Edges will be labeled with the guard of the transition, followed by the output, and a vector of Booleans to indicate which variables insample is stored in.

The working of $\mathcal{A}_{\mathsf{range}}$ in Fig. 1 can be explained as follows. Since insample′ is not output in any step, the parameters associated with sampling insample′ are not reported. The thresholds T_ℓ and T_u are hard-coded as 0 and 1, respectively, as the distribution means for the non-input states q_0 and q_1. The transition from q_0 to q_1 perturbs T_ℓ (= 0) and sets this to variable x_1; thus, x_1 corresponds to the variable low in Algorithm 1. The transition from q_1 to q_2 perturbs

[4] We could also allow guards of the form $\mathsf{insample} > \mathsf{x}_i$ and $\mathsf{insample} \leq \mathsf{x}_i$. However, we chose to keep the presentation simple. As all random variables in a DiPA are noisy, the equality happens with probability 0.

T_u (= 1) and stores it in x_2. Thus, x_2 corresponds to variable high in Algorithm 1. State q_2 is an input state. Transitions from q_2 perturb the query answer given as input storing it in insample, compare insample to the values stored in x_1 and x_2, and output the right value accordingly. State q_3 is a halting state where no transitions are enabled.

We conclude this example by illustrating the definitions associated with transitions. The transition t from q_0 to q_1 can be denoted by the quintuple $(q_0, \text{true}, q_1, \bot, (\text{true}, \text{false}))$. For t, we have $\text{src}(t) = q_0$, $\text{trg}(t) = q_1$, $\text{out}(t) = \bot$, $\text{guard}(t) = \text{true}$, $\text{smallv}(t) = \text{largev}(t) = \text{usedv}(t) = \emptyset$, $\text{assignv}(t) = \{\mathsf{x}_1\}$, and $\text{nonassignv}(t) = \{\mathsf{x}_2\}$. In this case t is a non-input, assignment transition for variable x_1. In contrast, the transition t' from q_2 to itself, is an input transition that is not an assignment transition for any variable. Here we have $\text{smallv}(t') = \{\mathsf{x}_1\}$, $\text{largev}(t') = \{\mathsf{x}_2\}$, and $\text{usedv}(t') = \{\mathsf{x}_1, \mathsf{x}_2\}$.

3.2 Semantics

An *execution/run* of a DiPA $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$, $\rho = t_0 t_1 \cdots t_{n-1}$, is a sequence of transitions t_i such that for every $0 < i < n$, $\text{trg}(t_{i-1}) = \text{src}(t_i)$ (i.e., the sequence ρ corresponds to a path in the "graph" of $\mathcal{A}$). We extend the notation of length, the ith transition, sub-sequence and suffix from (general) sequences: thus, $|\rho| = n$, $\rho[i] = t_i$, $\rho[i:j] = t_i \cdots t_{j-1}$ and $\rho[j:] = t_j t_{j+1} \cdots t_{n-1}$. We also extend the notion for source and target from transitions to a run — $\text{src}(\rho) = \text{src}(t_0)$ and $\text{trg}(\rho) = \text{trg}(t_{n-1})$. Using the notation developed for transitions, $\text{guard}(\rho[i])$ is the guard of the ith transition t_i of ρ. A run ρ is a *cycle* if $\text{src}(\rho) = \text{trg}(\rho)$, i.e., the run begins and ends in the same state. Finally, given two runs ρ_1 and ρ_2 such that $\text{trg}(\rho_1) = \text{src}(\rho_2)$, $\rho_1\rho_2$ is the run which is the concatenation of ρ_1 followed by ρ_2.

Recall that transitions of DiPA $\mathcal{A}$ compare values stored in the variables x_i $(1 \le i \le k)$ and insample. Thus, to define the semantics of the DiPA, we need to make sure that the value of variable x_i is defined before it is used in a comparison in the guard of a transition. Therefore, we make the technical assumption that on every run starting from the initial state q_{init}, a variable is assigned a value before it is referenced in a guard. We assume that all DiPA $\mathcal{A}$ considered in this paper are *initialized* as defined formally below.

Initialization: We say that a DiPA $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$ is *initialized* if for any run ρ starting from the initial state q_{init} (i.e., $\text{src}(\rho) = q_{\text{init}}$), if $\text{guard}(\rho[i])$ references variable x_ℓ (i.e., $\mathsf{x}_\ell \in \text{usedv}(\rho[i])$) then there is $j < i$ such that $\rho[j]$ is an assignment transition for x_ℓ (i.e., $\mathsf{x}_\ell \in \text{assignv}(\rho[j])$).

We need to define one more concept associated with a run ρ. For any storage variable x and position $j \in \{0, 1, \ldots \mid \rho|\}$, the *last position* when x was assigned before j is the maximum index $i < j$ such that x was assigned on transition $\rho[i]$. More precisely,

$$\text{lastassign}_\rho(\mathsf{x}, j) = \max\{i \mid i < j,\ \mathsf{x} \in \text{assignv}(\rho[i])\}.\ ^{5}$$

When the run ρ is clear from the context, we will drop the subscript and simply refer to the last assigned position before j for x as $\text{lastassign}(\mathsf{x}, j)$.

To define the semantics of a DiPA $\mathcal{A}$, we need to define the probability of "executions". But runs, as defined above, do not have all the information we need. For example, the real numbers read as input determine the values of insample and insample′, which in turn determine whether a transition is enabled and what is stored in the variables. Next, on transitions where either insample or insample′ are output, to define a meaningful measure space, we need to associate an interval (v, w) in which the output value lies. Thus, we define when a run corresponds to a certain sequence of inputs and outputs.

Definition 4 (Computation)**.** Consider DiPA $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$ and a run ρ of $\mathcal{A}$. Let $\sigma \in (\mathbb{R} \cup \{\tau\})^*$ be an *input sequence* and $\gamma \in (\Gamma \cup (\mathbb{R}_\infty \times \mathbb{R}_\infty))^*$ be an *output sequence*. We say that ρ is a *run on σ producing output γ* if the following conditions hold.

1. $|\rho| = |\sigma| = |\gamma|$.
2. For any i, $\sigma[i] = \tau$ iff $\text{src}(\rho[i]) \in Q_{\text{non}}$. That is, symbol τ is only read in non-input states.
3. For any i, $\gamma[i] \in \Gamma$ iff $\text{out}(\rho[i]) \in \Gamma$. Further for such i, $\text{out}(\rho[i]) = \gamma[i]$. That is, outputs in ρ "match" outputs in γ, with the only difference being that when insample or insample′ is output in ρ, the corresponding position in γ is an interval $(v, w) \in \mathbb{R}^2_\infty$.

When ρ is a run on σ producing γ, the tuple $\kappa = (\rho, \sigma, \gamma)$ will be called a *computation*.

For a computation $\kappa = (\rho, \sigma, \gamma)$ of DiPA $\mathcal{A}$, the suffix starting at position j is $\kappa[j:] = (\rho[j:], \sigma[j:], \gamma[j:])$. Notice that $\kappa[j:]$ (for any j) is also a computation of $\mathcal{A}$ since $\rho[j:]$ is a run on $\sigma[j:]$ producing $\gamma[j:]$. Also, we use length of κ, $|\kappa|$ to be $|\rho|$ $(= |\sigma| = |\gamma|)$, the length of the run ρ.

Probability of Computations. We will now define what the probability of each computation is. Recall that in each step, the automaton samples two values from Laplace distributions, and if the transition is from an input state, it adds the read input value to the sampled values and compares the result with the values stored in the variables x_i, $1 \le i \le k$. The step also outputs a value, and if the value output is one of the two sampled values, the computation requires it to belong to the interval that appears in the output sequence. The probability of such a transition thus is the probability of drawing a sample that satisfies the guard of the transition and (if the output is a real value) producing a number that lies in the interval in the output label. This intuition is formalized in a precise definition.

Let us fix a computation $\kappa = (\rho, \sigma, \gamma)$ of DiPA $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$. Recall that $\text{stor} = \{\mathsf{x}_i \mid 1 \le i \le k\}$. Since the parameters of the Laplace distribution that is used to sample insample and insample′ depend on the privacy budget ϵ, the probability of κ will also depend on ϵ. In addition, the values stored in the variables $\mathsf{x}_i \in \text{stor}$ at the start of the computation also influence the behavior of $\mathcal{A}$. Let $\eta : \text{stor} \to \mathbb{R}$ be the *evaluation* that defines the values of x_i, $1 \le i \le k$, initially. The probability of κ depends on both ϵ and η and is denoted as $\Pr[\epsilon, \eta, \kappa]$. We define this inductively on $|\kappa|$. For any ϵ and any computation κ with $|\kappa| = 0$, $\Pr[\epsilon, \eta, \kappa] = 1$.

Let us now consider the case when $|\kappa| > 0$. Before defining the probability in this case, we define the parameters that we will need. Let $P(\text{src}(\kappa[0])) = (d, \mu, d', \mu')$. Define the value a_0 as follows — if $\sigma[0] \in \mathbb{R}$ then $a_0 = \sigma[0]$, and if $\sigma[0] = \tau$ then $a_0 = 0$. Next, let us define the values ℓ and u. If $\gamma[0] \in \Gamma$ then $\ell = -\infty$ and $u = \infty$.

[5] As always $\max \emptyset = -\infty$ and $\min \emptyset = \infty$.

Otherwise, if $\gamma[0] = (v, w)$ then $\ell = v$ and $u = w$. Finally, for a parameter z, let η_z be the evaluation that modifies η by setting all the variables assigned by $\rho[0]$ to z. In other words,

$$\eta_z(\mathsf{x}) = \begin{cases} \eta(\mathsf{x}) & \text{if } \mathsf{x} \in \mathsf{nonassignv}(\rho[0]) \\ z & \text{if } \mathsf{x} \in \mathsf{assignv}(\rho[0]) \end{cases}$$

We are now ready to define $\Pr[\epsilon, \eta, \kappa]$ based on whether $\mathsf{out}(\rho[0]) = \mathsf{insample}'$ or not.

Case $\mathsf{out}(\rho[0]) = \mathsf{insample}'$**:** Set $\ell' = \max\{\eta(\mathsf{x}) \mid \mathsf{x} \in \mathsf{smallv}(\rho[0])\}$ and $u' = \min\{\eta(\mathsf{x}) \mid \mathsf{x} \in \mathsf{largev}(\rho[0])\}$. Also define p to be the probability that $\mathsf{insample}' \in (\ell, u) = (v, w) = \gamma[0]$, i.e.,

$$p = \int_{\ell}^{u} \frac{d'\epsilon}{2} e^{-d'\epsilon|z-\mu'-a_0|} dz$$

Then,

$$\Pr[\epsilon, \eta, \kappa] = p \int_{\ell'}^{u'} \left(\frac{d\epsilon}{2} e^{-d\epsilon|z-\mu-a_0|} \right) \Pr[\epsilon, \eta_z, \kappa[1:]] dz.$$

Case $\mathsf{out}(\rho[0]) \neq \mathsf{insample}'$**:** In other words, either $\mathsf{out}(\rho[0]) \in \Gamma$ or $\mathsf{out}(\rho[0]) = \mathsf{insample}$. In this case set $\ell' = \max(\{\eta(\mathsf{x}) \mid \mathsf{x} \in \mathsf{smallv}(\rho[0])\} \cup \{\ell\})$ and $u' = \min(\{\eta(\mathsf{x}) \mid \mathsf{x} \in \mathsf{largev}(\rho[0])\} \cup \{u\})$.

$$\Pr[\epsilon, \eta, \kappa] = \int_{\ell'}^{u'} \left(\frac{d\epsilon}{2} e^{-d\epsilon|z-\mu-a_0|} \right) \Pr[\epsilon, \eta_z, \kappa[1:]] dz.$$

In the special case when $\mathsf{assignv}(\rho[0]) = \emptyset$ (i.e., the first transition of the run does not change the assignment to any variable), observe that $\eta_z = \eta$. Hence, $\Pr[\epsilon, \eta_z, \kappa[1:]]$-term on the right hand side of both equations can be pulled out of the integral, and the expression can be simplified. We will abuse notation and use $\Pr[\cdot]$ to also refer to the function $\Pr[\eta, \kappa] := \epsilon \mapsto \Pr[\epsilon, \eta, \kappa]$. Notice that when ρ starts from q_{init}, because of the initialization condition of DiPA, the value of $\Pr[\cdot]$ does not depend on the valuation η. For such computations, we may drop the valuation η from the argument list of $\Pr[\cdot]$ to reduce notational overhead. Even though we plan to use the same function name, the number of arguments to $\Pr[\cdot]$ will disambiguate what we mean.

In this paper we study the computational problem of checking differential privacy for DiPAs. We conclude with a precise definition of this problem. We start by specializing the definition of differential privacy to the setting of DiPA. For a DiPA $\mathcal{A}$, an input sequence $\sigma \in (\mathbb{R} \cup \{\tau\})^*$ and an output sequence $\gamma \in (\Gamma \cup (\mathbb{R}_\infty \times \mathbb{R}_\infty))^*$, let $\mathsf{Runs}(\sigma, \gamma)$ be the set of all runs ρ of $\mathcal{A}$ starting from the initial state q_{init} such that ρ is a run on σ producing γ.

Definition 5. A DiPA $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private (for $\mathfrak{D} > 0$, $\epsilon > 0$) iff for every $\sigma_1, \sigma_2 \in (\mathbb{R} \cup \{\tau\})^*$ and $\gamma \in (\Gamma \cup (\mathbb{R}_\infty \times \mathbb{R}_\infty))^*$ such that σ_1 and σ_2 are *adjacent* [6],

$$\sum_{\rho \in \mathsf{Runs}(\sigma_1, \gamma)} \Pr[\epsilon, (\rho, \sigma_1, \gamma)] \leq e^{\mathfrak{D}\epsilon} \sum_{\rho \in \mathsf{Runs}(\sigma_2, \gamma)} \Pr[\epsilon, (\rho, \sigma_2, \gamma)].$$

Differential Privacy Problem: A DiPA $\mathcal{A}$ is said to be *differentially private* if there exists a constant $\mathfrak{D} > 0$ (independent of ϵ) such that $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private, $\forall \epsilon > 0$. The differential privacy problem is the problem of determining if a given DiPA $\mathcal{A}$ is differentially private.

[6] See Definition 1 on Page 3.

Remark. A DiPA $\mathcal{A}$ is a parametric automaton (with parameter ϵ), and the probability of each of its executions on a sequence of input varies with ϵ. Thus, considering its semantics, using $\mathcal{A}(\epsilon)$ to refer to the automaton may be more appropriate. However, we shall use $\mathcal{A}$ to reduce the notational overhead.

4 WELL FORMED DIPA

The main goal of the paper is to solve the differential privacy problem described in Section 3: Given a DiPA $\mathcal{A}$ determine if there is a $\mathfrak{D} > 0$ such that for all $\epsilon > 0$, $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private. In this section, we define the sub-class of *well-formed* DiPA that help characterize precisely the class of DiPA that are differentially private. Well-formed DiPA are automata that don't have four properties that lead to the violation of privacy: (a) *leaking cycles*, (b) *leaking pairs*, (c) *disclosing cycles*, and (d) *privacy violating paths*. We will define what these types of cycles and paths are in this section.

Dependency Graph of a Run. Consider a run ρ of a DiPA $\mathcal{A}$. Guards on transitions and decisions to store insample in storage variables, demand that if $\mathcal{A}$ follows the run ρ, then the values sampled as insample at different steps must be ordered in a certain way to ensure that guards are satisfied. This partial order on the sampled values demanded by a run is conveniently captured as a directed graph that we call the *dependency graph*.

Definition 6 (Dependency Graph)**.** Let $\mathcal{A} = (Q, \Gamma, q_{\mathsf{init}}, X, P, \delta)$ be a DiPA and let $\rho = t_0 t_1 \cdots t_{n-1}$ be a run of $\mathcal{A}$. The *dependency graph* of ρ is the directed graph $G_\rho = (V, E)$ where

- $V = \{i \mid 0 \leq i < n\}$, and
- E is defined as $E' \cap (V \times V)$ where

$$E' = \{(j, \mathsf{lastassign}_\rho(\mathsf{x}, j)) \mid j \in V,\ \mathsf{x} \in \mathsf{largev}(t_j)\} \cup \{(\mathsf{lastassign}_\rho(\mathsf{x}, j), j) \mid j \in V,\ \mathsf{x} \in \mathsf{smallv}(t_j)\}.$$

Notice that $E = E' \cap (V \times V)$ ensures that an edge $(j, \mathsf{lastassign}_\rho(\mathsf{x}, j))$ (or $(\mathsf{lastassign}_\rho(\mathsf{x}, j), j)$) is present only when $\mathsf{lastassign}_\rho(\mathsf{x}, j) \neq -\infty$ (i.e., when x is assigned before position j). Also observe that an edge (i, j) in G_ρ means that, to satisfy the guards, insample at position i in the run ρ must be less than insample at position j.

Given the intuition that the dependency graph G_ρ captures the ordering constraints imposed by the guards in ρ, one can conclude that a cycle in G_ρ means that ρ places contradictory demands on the values sampled and is therefore not a valid execution of the DiPA. We define a run ρ of DiPA $\mathcal{A}$ to be *feasible* iff G_ρ is acyclic. Feasibility is consistent with our semantic intuitions — if ρ is feasible then there is some evaluation η such that for any $\epsilon > 0$, any input sequence σ and any output sequence γ in which all output intervals are given by the interval $(-\infty, \infty)$, for which ρ is a run on σ that produces γ, $\Pr[\epsilon, \eta, (\rho, \sigma, \gamma)] > 0$.

Let us consider a feasible run $\rho = t_0 t_1 \cdots t_{n-1}$ of DiPA $\mathcal{A}$. Let $q_i = \mathsf{src}(t_i)$ and let $P(\mathsf{src}(t_i)) = (d_i, \mu_i, d_i', \mu_i')$. We say that ρ is *strongly feasible* if in addition whenever there is a path from i to j in G_ρ and $q_i, q_j \in Q_{\mathsf{non}}$ then $\mu_i < \mu_j$. Thus, ρ is strongly feasible if whenever guards require two insample values on non-input transitions to be ordered, the corresponding means of the Laplace distributions are ordered in the same way. We only consider DiPA that satisfy the following *strong feasibility assumption*.

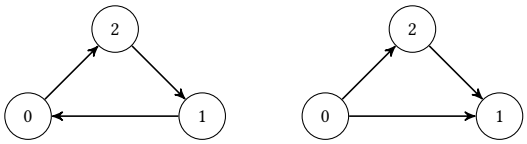

Figure 2: **Dependency graphs for runs ρ_1 and ρ_2 from Example 3. G_{ρ_1} is on the left and G_{ρ_2} is on the right.**

Strong Feasibility: All feasible runs from the initial state q_{init} are strongly feasible.

Example 3. Let us look at two example runs of length 3.

$\rho_1 = (q_0, \text{true}, q_1, \bot, (\text{true}, \text{false}))(q_1, \text{insample} < \mathsf{x}_1, q_2, \bot, (\text{false}, \text{true}))$
$(q_2, \text{insample} \geq \mathsf{x}_1 \wedge \text{insample} < \mathsf{x}_2, q_3, \bot, (\text{false}, \text{false}))$

$\rho_2 = (q_0, \text{true}, q_1, \bot, (\text{true}, \text{false}))(q_1, \text{insample} \geq \mathsf{x}_1, q_2, \bot, (\text{false}, \text{true}))$
$(q_2, \text{insample} \geq \mathsf{x}_1 \wedge \text{insample} < \mathsf{x}_2, q_3, \bot, (\text{false}, \text{false}))$

The only difference between ρ_1 and ρ_2 is the guard on the second transition, which goes from state q_1 to q_2. Their dependency graphs are shown in Figure 2. G_{ρ_1} is on the left and can be explained as follows. Transition 0 sets variable x_1 and transition 1 sets variable x_2. The guard $\text{insample} < \mathsf{x}_1$ in transition 1 results in the edge from 1 to 0. The conjunct $\text{insample} \geq \mathsf{x}_1$ in transition 2 results in an edge from 0 to 2, and the conjunct $\text{insample} < \mathsf{x}_2$ results in the edge from 2 to 1. G_{ρ_1} is cyclic which means that ρ_1 is not feasible. Graph G_{ρ_2} on the right in Figure 2 is similar but the guard $\text{insample} \geq \mathsf{x}_1$ in transition 1 results in an edge from 0 to 1 (instead of from 1 to 0 in G_{ρ_2}) which removes the cycle. Thus, ρ_2 is feasible.

Leaking cycle. We are now ready to present the first graph theoretic condition on DiPA that demonstrates a violation of differential privacy.

Definition 7 (Leaking cycle)**.** A run ρ of $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$ from the initial state q_{init} (i.e., $\text{src}(\rho) = q_{\text{init}}$) is said to be a *leaking cycle* if there is an index $0 \leq j < |\rho|$ and a storage variable $\mathsf{x} \in \text{stor}$ such that the following conditions hold.

Cycle: $C = \rho[j:]$ is a cycle.

Leak: There are indices i_1 and i_2 in C (i.e., $j \leq i_1, i_2$) such that $\mathsf{x} \in \text{assignv}(\rho[i_1])$ and $\mathsf{x} \in \text{usedv}(\rho[i_2])$.

Repeatability: C can be repeated arbitrarily many times. That is, for every $m \geq 0$, the run ρC^m is feasible. [7]

Intuitively, the condition Leak in Definition 7 is to ensure that variable x is assigned a value in the cycle C that is later tested against in a guard. [8] The main effect of the 3 conditions in Definition 7, is to identify two transitions (namely, those corresponding to assignment and test) that can be taken arbitrarily many times (since they are on a repeatable cycle) such that the insample values sampled in the two transitions are ordered in the same way each time the transitions are taken. This property leads to a "leaking" of the privacy budget, as shall be explained when we sketch the proof.

A cycle C that does not satisfy the condition Leak will be said to be *non-leaking*.

[7] C^m denotes the m-fold concatenation of C with $C^0 = \lambda$.

[8] Definition 7 does not require $i_1 < i_2$. Therefore, strictly speaking the assignment in i_1 may not be before the test in i_2. But this can be easily addressed by taking C^2 instead of C as the cycle.

Definition 8 (Non-leaking cycle)**.** A run C is a *non-leaking cycle* if C is a cycle and for every $\mathsf{x} \in \text{stor}$ and i, if $\mathsf{x} \in \text{usedv}(C^2[i])$ then $\text{lastassign}_{C^2}(\mathsf{x}, i) = -\infty$, i.e, x is not assigned a value in C. Here C^2 is the concatenation of C with itself.

In Definition 8, we use the run C^2 to ensure that we also account for the case when x is assigned *after* it is used in C. One important property about non-leaking cycle is that it is always repeatable; this is the content of the next proposition. Thus repeatability is a non-trivial requirement only for cycles that have a leak.

Proposition 1. *Let ρ be a feasible run of $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$ from the initial state q_{init} such that $C = \rho[i:j]$ (for some $0 \leq i < j \leq |\rho|$) is a non-leaking cycle. Then for every $m > 0$, $\rho[0:i](\rho[i:j])^m \rho[j:]$ is feasible.*

Leaking pair. Recall that the key property of a leaking cycle that leads to the violation of differential privacy is finding two transitions that can be repeated arbitrarily many times such that the insample value sampled in the two transitions is ordered every time they are taken. Leaking cycles achieve this by finding both transitions on a cycle that can be repeated. However, that is not the only way such a pair of transitions can arise — the two transitions could be on two different cycles that can each be repeated. This leads to the definition of a *leaking pair*. The definition of a leaking pair is subtle and we will discuss its details after presenting it formally.

Definition 9 (Leaking pair)**.** A feasible run ρ of $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$ from the initial state q_{init} is a *leaking pair* if there are indices $0 \leq i_1 < j_1 \leq |\rho|$ and $0 \leq i_2 < j_2 \leq |\rho|$ such that the following conditions hold.

Cycles: $C_1 = \rho[i_1 : j_1]$ and $C_2 = \rho[i_2 : j_2]$ are both non-leaking cycles.

Disjointness: Either $j_1 \leq i_2$ or $j_2 \leq i_1$. That is, C_1 and C_2 are non-overlapping subsequences of ρ.

Order: There is a path $k_1, k_2, \ldots k_m$ in the dependency graph G_ρ such that $i_1 \leq k_1 < j_1$ (k_1 is on C_1), $i_2 \leq k_m < j_2$ (k_m is on C_2), $k_2 < k_1$ and $k_{m-1} < k_m$.

As mentioned before Definition 9, the motivation behind leaking pairs is to identify a pair of transitions t and t' that can be executed multiple times and such that the insample value each time t is taken is smaller than the insample value each time t' is taken. Such a pair of transitions represents a "leak" of the privacy budget that can be exploited to prove that DiPA is not differentially private. Definition 9 achieves this goal in the following manner. The desired transitions t and t' are $\rho[k_1]$ and $\rho[k_m]$, respectively. The fact that t and t' are on cycles C_1 and C_2 which are disjoint (in ρ) and non-leaking, ensures that they can be repeated thanks to Proposition 1. The condition Order in Definition 9 is the most subtle. The fact that $k_2 < k_1$ and (k_1, k_2) is an edge in G_ρ means that there is a storage variable $\mathsf{x} \in \text{stor}$ such that x is assigned in $\rho[k_2]$ and $\text{insample} < \mathsf{x}$ is one of the conjuncts in $\text{guard}(\rho[k_1])$. Further since C_1 is non-leaking, x is not updated within C_1 and hence $\rho[k_2]$ is taken *before* C_1. Similar conclusions can be drawn about k_{m-1} and k_m — there is a variable $\mathsf{y} \in \text{stor}$ that is assigned in $\rho[k_{m-1}]$ which is taken before C_2, and $\text{insample} \geq \mathsf{y}$ is a conjunct in $\text{guard}(\rho[k_m])$. Finally, the path from k_1 to k_m means that the insample value sampled in $\rho[k_1]$ is less than the value assigned to x in $\rho[k_2]$, which in turn is

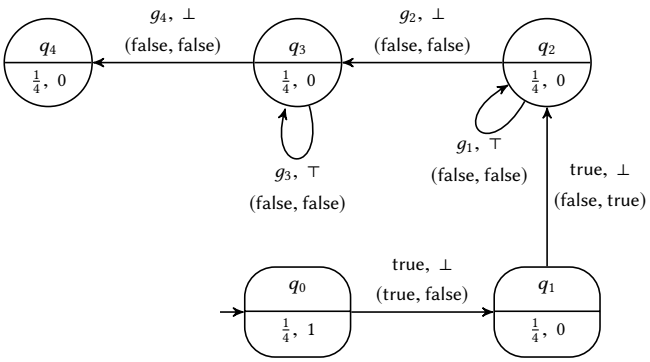

Figure 3: **DiPA $\mathcal{A}_{\mathsf{leakp}}$ from Example 4. $\mathcal{A}_{\mathsf{leakp}}$ has two variables, x_1 and x_2, assigned in the first and the second transition, respectively. The guards** $g_1 = (\mathsf{insample} \geq \mathsf{x}_1)$, $g_2 = (\mathsf{insample} < \mathsf{x}_1)$, $g_3 = (\mathsf{insample} < \mathsf{x}_2)$, $g_4 = (\mathsf{insample} < \mathsf{x}_1) \wedge (\mathsf{insample} \geq \mathsf{x}_2)$.

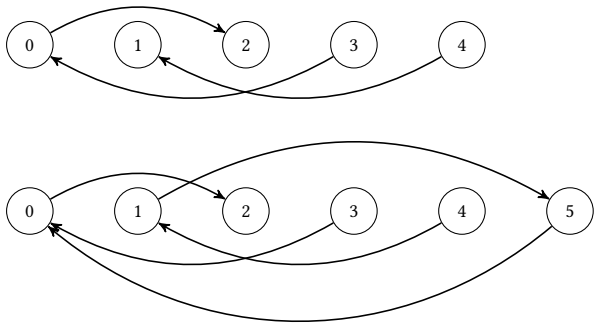

Figure 4: **Dependency graphs for runs ρ_1 and ρ_2 from Example 4. G_{ρ_1} is on the top and G_{ρ_2} is on the bottom. The nodes are numbered according to the order in which the corresponding transition appears in the run.**

less than the value assigned to y in $\rho[k_{m-1}]$ and that is less than the insample value sampled in $\rho[k_m]$. $\rho[k_2]$ is before C_1 which means that the value assigned to x in $\rho[k_2]$ does not change no matter how many times C_1 and C_2 are repeated. Next, $\rho[k_{m-1}]$ is before C_2. It is possible that $\rho[k_{m-1}]$ is on C_1, in which case the value assigned to y changes when C_1 is repeated. However, one can show by induction, that the presence of a path in the dependency graph from $\rho[k_2]$ to $\rho[k_{m-1}]$ and an edge from $\rho[k_{m-1}]$ to $\rho[k_m]$ means that when C_1 and C_2 are repeated, there will be a path from $\rho[k_2]$ and the last instance of $\rho[k_{m-1}]$ and the last value assigned to y in $\rho[k_{m-1}]$ will be less than every insample value sampled in $\rho[k_m]$. Thus, every insample value sampled in $\rho[k_1]$ will be less than every insample value sampled in $\rho[k_m]$, no matter how many times C_1 and C_2 are repeated.

Example 4. Consider the automaton $\mathcal{A}_{\mathsf{leakp}}$ in Figure 3. The automaton is drawn following the convention outlined in Example 2. The automaton has two real variables x_1 and x_2, assigned in the first and the second transition, respectively. For states q_i, q_j of $\mathcal{A}_{\mathsf{leakp}}$, let t_{ij} denote the unique transition of $\mathcal{A}_{\mathsf{leakp}}$ from state q_i to q_j. Observe that t_{22} and t_{33} are cycles. Consider the run $\rho_1 = t_{01}t_{12}t_{22}t_{23}t_{33}$ that visits both the cycles t_{22} and t_{33} and its extension $\rho_2 = \rho_1 t_{34}$. Their dependency graphs for these runs are shown in Figure 4. The nodes 2 and 4 correspond to the cycle transitions t_{22} and t_{33} respectively. Considering just the run ρ_1, these cycles do not constitute a leaking pair. However, when we consider the extended run, ρ_2, we see that these cycles form a leaking pair via the path $4 \longrightarrow 1 \longrightarrow 5 \longrightarrow 0 \longrightarrow 2$.

Before moving onto the other two properties needed to define well-formed DiPA, it is useful to remark that the cycles C_1 and C_2 in Definition 9 may be the "same cycle", i.e., C_1 and C_2 could, respectively, be the first and second iterations of the same sequence of $\mathcal{A}$ transitions.

Disclosing cycle. Real valued outputs present another avenue through which private information in the input can be leaked. The condition identified by leaking cycles and leaking pairs do not account for such violations because they are agnostic to the type of output produced by the DiPA. Our next condition *disclosing cycle*, identifies a transition that can be executed repeatedly, and which outputs a pertubed input.

Definition 10 (Disclosing cycle)**.** A feasible run ρ of $\mathcal{A} = (Q, \Gamma, q_{\mathsf{init}}, X, P, \delta)$ from the initial state q_{init} is a *disclosing cycle* if there are indices $0 \leq j \leq i < |\rho|$ such that the following conditions hold.

Cycle: $C = \rho[j:]$ is a non-leaking cycle.
Disclosing: $\rho[i]$ is an input transition that outputs a real value, i.e., $\mathsf{src}(\rho[i]) \in Q_{\mathsf{in}}$ with $\mathsf{out}(\rho[i]) \in \{\mathsf{insample}, \mathsf{insample}'\}$.

Observe that in Definition 10, $\rho[i]$ is a transition that is on cycle C. Moreover, since C is non-leaking cycle, by Proposition 1, the run ρC^m is feasible for every $m \geq 0$. Thus, the transition $\rho[i]$ can be executed repeatedly. Since $\rho[i]$ is an input transition that outputs a real-value, each time it is executed it reveals some information about the input which results in a loss of privacy.

Privacy violating path. We now present the last property needed to define well formed DiPA. This last property also concerns privacy violations that arise from real valued outputs. Leaking cycles and leaking pairs identify a transition that is executed arbitrarily many times where the sampled insample value is bounded by values sampled in another transition (that is also executed many times) on the same run. However, with real valued outputs, we could have a situation where this bound is revealed once, explicitly in an output. This is captured in our next definition.

Definition 11 (Privacy violating path)**.** A feasible run ρ of $\mathcal{A} = (Q, \Gamma, q_{\mathsf{init}}, X, P, \delta)$ from the initial state q_{init} is a *privacy violating path* if there are indices $0 \leq i \leq j \leq |\rho|$ such that the following conditions hold.

Cycle: $C = \rho[i:j]$ is a non-leaking cycle.
Privacy Violation: There is a path $k_1, k_2, \ldots k_m$ in the dependency graph G_ρ such that either (a) $\mathsf{out}(\rho[k_1]) = \mathsf{insample}$, $k_{m-1} < k_m$, and $i \leq k_m < j$, i.e., $\rho[k_m]$ is on cycle C, or (b) $i \leq k_1 < j$ ($\rho[k_1]$ is on cycle C), $k_2 < k_1$, and $\mathsf{out}(\rho[k_m]) = \mathsf{insample}$.

It is useful to see how Definition 11 captures the intuitions laid out before. The path from k_1 to k_m in G_ρ ensures that the insample

value sampled in $\rho[k_1]$ is less than the insample value sampled in $\rho[k_m]$. Moreover, since C is non-leaking, by Proposition 1, it is repeatable. Condition (a) in (Privacy Violation) says that $\rho[k_m]$ is a transition on C, and the edge (k_{m-1}, k_m) in G_ρ along with $k_{m-1} < k_m$ means that there is a variable $\mathsf{x} \in$ stor that is set in $\rho[k_{m-1}]$ and insample $\geq \mathsf{x}$ is in guard($\rho[k_m]$). Moreover, since C is non-leaking, x is not updated in C and hence k_{m-1} is before C. Thus, the presence of the path means that the value output in $\rho[k_1]$ is less than the insample value sampled in $\rho[k_{m-1}]$ which in turn is less than the insample value sampled in $\rho[k_m]$ every time C is repeated. Therefore, there is a lower bound, which is output in $\rho[k_1]$, for arbitrary many insample values that are generated in $\rho[k_m]$. Condition (b) in (Privacy Violation) is similar but *dual*. Here $\rho[k_1]$ is on C, $\rho[k_2]$ is before C and sets a variable x that is an upper bound on the values sampled in $\rho[k_1]$, and finally, $\rho[k_m]$ outputs a value that upper bounds all these values, no matter how many times $\rho[k_1]$ is executed by repeating C.

Well-formed DiPA. The properties defined in this section identify witnesses for the violation of privacy. The class of *well-formed* automata are those that do not suffer from these deficiencies.

Definition 12 (Well-formed DiPA)**.** A DiPA $\mathcal{A}$ is said to be *well-formed* if $\mathcal{A}$ does not have any leaking cycles, leaking pairs, disclosing cycles, and privacy violating paths.

Our main results are: (i) a well-formed DiPA is differentially private; (ii) if a DiPA satisfying the output distinction property (see Definition 13) is differentially private then it must be well-formed. We will also show that there is an effective procedure for checking if a DiPA is well-formed. These observations together will provide a decidability result for solving the differential privacy problem for DiPA that satisfy output distinction property.

5 WELL-FORMED DIPA ARE DIFFERENTIALLY PRIVATE

One of our main results, which we call the *sufficiency theorem*, is that well-formed DiPAs are differentially private.

Theorem 2. *Let $\mathcal{A}$ be a DiPA. If $\mathcal{A}$ is well-formed then there is a $\mathfrak{D} > 0$ such that for every $\epsilon > 0$, $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private. Further, such a $\mathfrak{D}$ can be computed in time exponential in the size of the automaton $\mathcal{A}$.*

Proof Sketch. Let $\mathcal{A}$ be a well-formed DiPA. Given a feasible run $\rho = t_0 \cdots t_n$ of $\mathcal{A}$ from the initial state, fix computations $\kappa_i = (\rho, \sigma_i, \gamma)$ for $i = 1, 2$ such that σ_1 and σ_2 are adjacent. For each j, let lt_j be the "less than" relation on stor imposed by the prefix $\rho[0 : j-1] - (\mathsf{x}, \mathsf{x}') \in \mathsf{lt}_j$ if there is a path of non-zero length from $\mathsf{lastassign}_\rho(\mathsf{x}, j)$ to $\mathsf{lastassign}_\rho(\mathsf{x}', j)$. Similarly, eq_j is the "equality" relation on stor imposed by the prefix $\rho[0 : j-1] - (\mathsf{x}, \mathsf{x}') \in \mathsf{eq}_j$ if $\mathsf{lastassign}_\rho(\mathsf{x}, j) = \mathsf{lastassign}_\rho(\mathsf{x}', j)$.

We can show that there are numbers wt_j and functions m_j : stor $\to \{-1, 0, 1\}$ such that

(1) For any valuations η_1, η_2 such that $\eta_2 = \eta_1 + m_j$, [9]

$$\mathsf{Prob}[\epsilon, \eta_2, \kappa_2[j:]] \leq e^{\sum_{\ell=j}^{n-1} \mathsf{wt}_\ell} \mathsf{Prob}[\epsilon, \eta_1, \kappa_1[j:]].$$

[9]For functions $f, g : A \to \mathbb{R}$, $f + g$ is the function that adds the result of f and g for each argument, i.e., $(f + g)(a) = f(a) + g(a)$.

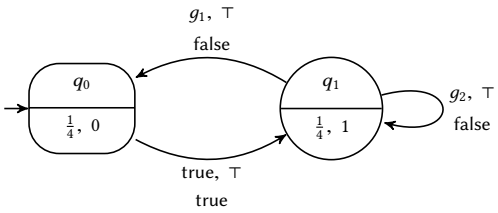

Figure 5: **DiPA $\mathcal{A}_{\mathsf{nwf}}$ with one variable x is not well-formed but differentially private. The guards g_1 = (insample $\geq$ x) and g_2 = (insample $<$ x).**

(2) If $t_{j_1} = t_{j_2}$, $\mathsf{lt}_{j_1} = \mathsf{lt}_{j_2}$ and $\mathsf{eq}_{j_1} = \mathsf{eq}_{j_2}$ for $j_1 \leq j_2$ then $\mathsf{wt}_{j_1} = 0$
(3) $\mathsf{wt}_j \leq 2d_j + d'_j$ where d_j and d'_j are such that $P(\mathsf{src}(t_j)) = (d_j, \mu_j, d'_j, \mu'_j)$.

Observe that the last two conditions imply that there is a number $\mathfrak{D}$ independent of ρ such that $\sum_{\ell=j}^{n-1} \mathsf{wt}_\ell < \mathfrak{D}$. Note that as ρ is a run from initial state then $\mathsf{Prob}[\epsilon, \eta_i, \kappa_i]$ is independent of η_i. The above observations imply that

$$\mathsf{Prob}[\epsilon, \kappa_2] \leq e^{\mathfrak{D}} \mathsf{Prob}[\epsilon, \kappa_1].$$

This shows that $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private. To carry out the formal proof, we construct an augmented automaton $\mathsf{aug}(\mathcal{A})$, whose states are triples of the form $(q, \mathsf{lt}, \mathsf{eq})$ where q is a state of $\mathcal{A}$, lt, and eq are strict partial orders and equivalence relations on stor. The value for $\mathfrak{D}$ is also computed using the augmented automaton. □

The problem of checking well-formedness can be shown to be in PSPACE.

Theorem 3. *The problem of checking whether a DiPA is well-formed is in PSPACE. When the number of variables is taken to be a constant k, then the problem of checking whether a DiPA is well-formed is decidable in polynomial time.*

6 DIFFERENTIALLY PRIVATE DIPA ARE WELL-FORMED

While well-formedness is sufficient for ensuring differential privacy, it is not a necessary condition for differential privacy as illustrated by the following example.

Example 5. Consider the DiPA $\mathcal{A}_{\mathsf{nwf}}$ with one variable insample given in Figure 5. The automaton is drawn following the convention outlined in Example 2. As each transition outputs ⊤, $\mathcal{A}_{\mathsf{nwf}}$, on any input of length n, outputs the string $\top^n$ with probability 1. Thus, $\mathcal{A}_{\mathsf{nwf}}$ is trivially differentially private. However, $\mathcal{A}_{\mathsf{nwf}}$ is not well-formed as it has a leaking cycle, $t_a t_b$ where t_a is the transition from q_0 to q_1 and t_b is the transition from q_1 to q_0.

We show, however, that differentially private DiPA that satisfy an additional technical property of *output distinction* are well-formed. Thus, for DiPA satisfying this property, well-formedness is a precise characterization of when they are differentially private. Before presenting this *restricted necessity* theorem and proof sketch, let us define what it means for a DiPA to satisfy the condition of output distinction.

Definition 13 (Output Distinction)**.** A DiPA $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$ satisfies *output distinction* if the following holds: If t_1 and t_2 are distinct transitions of $\mathcal{A}$ such that $\text{src}(t_1) = \text{src}(t_2)$ then $\text{out}(t_1) \neq \text{out}(t_2)$ and $\{\text{out}(t_1), \text{out}(t_2)\} \cap \Gamma \neq \emptyset$.

Output Distinction demands that distinct outgoing transitions from a state have different outputs and at most one of the outgoing transitions outputs a real value. In particular, there cannot be two transitions out of a state q that output insample and insample′. Distinct outputs on transitions ensure that given a starting state q and an output sequence γ, there is at most one run ρ starting from q that can produce γ. Observe that the automaton of Figure 5 does not satisfy output distinction property. The necessity proof proceeds by showing that if $\mathcal{A}$ is not well-formed, then given $\mathfrak{D}$, there are computations (ρ, σ_1, γ) and (ρ, σ_2, γ) with the same run ρ such that ρ outputs γ, σ_1, σ_2 are adjacent and the ratio of the probability measures of these computations is $> e^{\mathfrak{D}\epsilon}$ for sufficiently large ϵ. Output distinction guarantees that ρ is the *only* run on σ_1, σ_2 that outputs γ, allowing us to conclude that $\mathcal{A}$ is not differentially private for non-well formed $\mathcal{A}$. Without output distinction, the deficit in probability measures of γ can be made up by other paths. The output distinction property is also needed in [10] for the case of a single variable. We are now ready to present the main result of this section.

Theorem 4. *Let $\mathcal{A}$ be a DiPA that satisfies the output distinction property. If $\mathcal{A}$ is not well-formed, then it is not differentially private.*

Proof Sketch. Let us fix a DiPA $\mathcal{A} = (Q, \Gamma, q_{\text{init}}, X, P, \delta)$ that satisfies the output distinction property. Recall that the output distinction property ensures that for any input sequence σ and output sequence γ, $|\text{Runs}(\sigma, \gamma)| \leq 1$. We sketch the main ideas behind the proof; the full details can be found in [11]. Assume that $\mathcal{A}$ is not well-formed. Now, for each value of $\mathfrak{D}$ and ϵ, the proof identifies a run ρ, an output sequence γ, and a pair of adjacent input sequences α and β such that the computations (ρ, α, γ) and (ρ, β, γ) demonstrate a violation of differential privacy (Definition 5). The construction of witnesses is based on the following sequence of observations.

(1) Let us fix a run ρ from q_{init} and an output sequence γ consistent with ρ. Observe that the number read in an input transition determines the mean of the distributions from which insample and insample′ are drawn in that step. Let us call an input sequence σ *strongly compliant* with ρ and γ, if the sampling means satisfy the constraints imposed by ρ and γ. This has two requirements. First, whenever there is a path from i to j in G_ρ, the sample mean at step i is less than the sample mean at step j. Notice that strong feasibility ensures this when i and j are non-input transitions, and here we are requiring this to hold when either i or j is an input transition in which case the mean is determined by σ. Second, if $\text{out}(\rho[i]) \in \{\text{insample}, \text{insample}'\}$ (real outputs), the sample mean at step i is in the interval $\gamma[i]$. Intuitively, for a strongly compliant input sequence σ, the probability of computation (ρ, σ, γ) is likely to be "high". On the flip side, let us call an input sequence σ *non-compliant* at i, if the sample mean set by σ at step i either violates an order constraint or an output constraint. Again intuitively, one can imagine that, as the number of non-compliant transitions increase in σ, the probability of the computation (ρ, σ, γ) decreases. Now one can prove that if we consider two input sequences σ_1, which is strongly compliant, and σ_2, which has non-compliant transitions, then the ratio of the probabilities of (ρ, σ_1, γ) and (ρ, σ_2, γ) grows as the number of non-compliant transitions in (ρ, σ_2, γ) increases.

(2) Observations in (1) above provide a template for how to identify witnesses for differential privacy violation: the presence of a leaking cycle, leaking pair, disclosing cycle, or privacy violating path help identify a run, and we then construct two input sequences α, which is strongly compliant, and β which has many non-compliant steps. Observe that each witness to non-well-formedness is a run containing a cycle that can be repeated arbitrarily many times and contains a transition that will be made non-compliant in the input sequence β. The intuitions laid out in Section 4 for defining well-formed DiPA will be used and we spell this out in each case. A leaking cycle has a transition with index i_1 (see Definition 7) that sets a variable which is then used later in the transition indexed i_2. Since the guard of i_2 is not true, it is an input transition. We will construct the run ρ by repeating the cycle as many times as needed (based on $\mathfrak{D}$ and ϵ), and in β the sample mean at step i_2 will be in the wrong order with respect to i_1 in each repetition, making it non-compliant. In a leaking pair (Definition 9) there is a pair of transitions indexed k_1 and k_m on cycles that can be repeated, and whose sampled values need to be ordered each time they are executed. Moreover, transitions k_1 and k_m are input transitions because their guards are not true (see discussion after Definition 9). Thus, in β we will flip the order of the sample means at these steps to create an arbitrary number of non-compliant steps. The transition indexed i in a disclosing cycle (Definition 10) is an input transition on a cycle that can be repeated. To create non-compliant steps in β we will set the mean of these transitions to not be in the output interval given for this step. Finally, in a privacy violating path (Definition 11) there is an input transition with index k_m for case (a) (or k_1 for case (b)) that is on a repeatable cycle whose sampled value is required to be larger than (smaller than in case (b)) the value output in step k_1 (step k_m for case (b)). To construct the input sequence β, we set the input for each time k_m (k_1 in case (b)) is taken to be smaller than the value output in k_1, and thereby creating arbitrarily many non-compliant steps.

(3) The general principles behind constructing the input sequences α and β are laid out in (2). However, one key requirement for α and β to constitute a witness to privacy violation is that they be *adjacent* (Definition 1) which demands that the values in α and β be not too far apart. One challenge is carrying this out is the presence of non input transitions, where the sample means are *fixed*. This can be overcome by carefully analyzing the dependency graph G_ρ and the parameters decorating the states appearing in the run ρ. □

In Section 5, we showed that there is a PSPACE algorithm to determine if an output-distinct DiPA $\mathcal{A}$ is well-formed (Theorem 3). This complexity bound is tight; we show that the problem of determining if a DiPA is differentially private is PSPACE-hard. (See [11] for the proof.)

Theorem 5. *Given an output-distinct DiPA $\mathcal{A}$, the problem of determining if there is a $\mathfrak{D} > 0$ such that for all ϵ, $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private, is PSPACE-hard.*

7 EXPERIMENTS

We implemented the algorithm that checks whether a DiPA $\mathcal{A}$ is well-formed. In case $\mathcal{A}$ is well-formed, it computes a bound $\mathfrak{D}$, which we call the *weight* of the automaton, such that $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private for all ϵ. The software tool, DiPAut, is built in Python 3.9.5 and is available for download at [7]. It uses the PLY package [6] for parsing the program and the IGRAPH package [14] to store the input automaton as a graph. The IGRAPH package is also used to perform graph-theoretic operations on the input automaton.

DiPAut has three major components. The first component, called *core*, tokenizes and parses the input using PLY. The second component, *builders*, constructs the augmentation of the input automaton. The augmentation is built using a breadth-first-search of the (implicit) graph of the augmentation. The relations lt and eq are stored as dictionaries during augmentation. To prepare for checking of leaking pair and privacy violating path, the automaton also builds an "enhanced" augmentation. For example, it also builds the graphs that include assignments to the variables V_1 and V_2 in the algorithm for checking leaking pair (See the proof of Theorem 3). The third component *DP tests*, implements the finals checks for leaking cycle, leaking pair, privacy violating path and disclosing cycle from the augmentations. If the automaton is well-formed, it also computes the weight of the automaton. If it is not well-formed, it further checks if it is output-distinct. In that case, we report that the automaton is not differentially private.

DiPAut was evaluated against a suite of examples (See Table 1), which we describe briefly.

7.1 Description of Examples

The first examples we consider are the standard Sparse Vector Technique (SVT) [19] and the Numeric Sparse (NUM-SPARSE) [20]. These algorithms use one variable. Detailed discussion of these algorithms can be found in [19, 20]. Apart from SVT and NUM-SPARSE, all other examples use more than one variable.

We also designed new examples, described below. The first set of examples was designed to ensure that the tests of well-formedness were implemented correctly. A second set of examples were designed to evaluate the scalability of our tool. They include k-MIN-MAX (for each $k > 0$) and m-RANGE (for each $m > 0$). The 1-RANGE is the range query algorithm given in Example 1.

Examples LC-EXAMPLE and DC-EXAMPLE. The algorithm LC-EXAMPLE and DC-EXAMPLE are variants of 1-RANGE. The algorithm LC-EXAMPLE is designed to have a leaking cycle and DC-EXAMPLE is designed to have a disclosing cycle. A detailed description of the algorithms can be found in [11].

Examples NUM-RANGE-1 and NUM-RANGE-2. The algorithm NUM-RANGE-1 is the variant of 1-Range which outputs insample (instead of $\top$) when the sampled value $q[i]$ is greater than high. The algorithm NUM-RANGE-2 on the other hand outputs insample$'$. NUM-RANGE-2 is well-formed, output-distinct and hence differentially private but NUM-RANGE-1 has a privacy-violating path. A detailed description of the algorithms can be found in [11].

Examples TWO-RANGE-1 and TWO-RANGE-2. TWO-RANGE-1 is a variant of 1-RANGE. In both algorithms, at the beginning, three thresholds, T_ℓ, T_m, and T_u, are perturbed by adding noise sampled from the Laplace distribution. The algorithms then proceed to process the queries, checking if the remaining noisy queries are between the noisy T_ℓ and T_m. If at some point the input noisy query exceeds the noisy T_m, TWO-RANGE-1 checks that the remaining queries are in between the noisy T_m and the noisy T_u. In contrast, the algorithm TWO-RANGE-2 resamples T_m before checking that the remaining queries are in between the noisy T_m and the noisy T_u. TWO-RANGE-1 has a leaking pair and is not differentially privacy. TWO-RANGE-2, on the other hand, is well-formed, output distinct, and hence differentially private TWO-RANGE-1 and TWO-RANGE-2 are described in detail in [11].

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

min, max $\leftarrow$ Lap($\frac{\epsilon}{4k}, q[1]$))
for $i \leftarrow 2$ **to** k **do**
  r $\leftarrow$ Lap($\frac{\epsilon}{4k}, q[i]$)
  if (r > max) $\wedge$ (r > min) **then**
    max $\leftarrow$ r
  else if (r < min) $\wedge$ (r < max) **then**
    min $\leftarrow$ r
  end
  $out[i] \leftarrow$ read
end
for $i \leftarrow k+1$ **to** N **do**
  r $\leftarrow$ Lap($\frac{\epsilon}{4}, q[i]$)
  if (r $\geq$ min) $\wedge$ (r < max) **then**
    $out[i] \leftarrow \perp$
  else if (r $\geq$ min) $\wedge$ (r $\geq$ max) **then**
    $out[i] \leftarrow \top$
    exit
  else if (r < min) $\wedge$ (r < max) **then**
    $out[i] \leftarrow \perp$
    exit
  end
end

Algorithm 2: k-MIN-MAX algorithm. k-MIN-MAX is differentially private.

Example k-MIN-MAX. One set of examples designed to check scalability of our algorithm is k-MIN-MAX ($k \geq 2$). Initially, k-MIN-MAX reads k-queries, adds noise from the Laplace distribution at each step, remembering the maximum and minimum amongst the perturbed queries. During this phase, the outputs do not inform the observer whether the noisy query being processed updates the maximum or minimum.

After reading the first k-queries, each subsequent query is perturbed by adding noise, and the algorithm checks if the noisy query is between the maximum and minimum found in the first k-noisy queries. It continues processing the queries as long as it is between those two. Otherwise, it quits. Observe that k-MIN-MAX is a parametric set of examples, one for each value of k. For each k, the DiPA modeling k-MIN-MAX has *two* variables, has $k+2$ states and $3k+1$ transitions. Further, k-MIN-MAX does not satisfy output distinction

Benchmarks				DiPAut				CheckDP [29]	
Example	v	s	$trans$	wt calc time (s)	total time (s)	differentially private?	$\mathfrak{D}$	time (s)	Counterexample Validated?
SVT	1	3	3	0.00046	0.238	✓	5/4	29.92	N.A.
Num-Sparse	1	3	3	0.00045	0.249	✓	7/4	52.43	N.A.
DC-Example	2	4	5	N.A.	0.237	×, DC	N.A.	43.59	T.O.
Num-Range-1	2	4	4	N.A.	0.234	×, PV	N.A.	316.05	T.O.
Num-Range-2	2	4	4	0.00078	0.231	✓	5/4	1909.43	T.O.
LC-Example	2	4	4	N.A.	0.231	×, LC	N.A.	T.O.	
Two-Range-1	3	6	10	N.A.	0.239	×, LP	N.A.	T.O.	
Two-Range-2	3	7	11	0.00258	0.277	✓	2	T.O.	
2-Min-Max	2	4	7	0.00065	0.220	✓	1	T.O.	
10-Min-Max	2	12	31	0.00221	0.230	✓	1	M.E.	
20-Min-Max	2	22	61	0.00434	0.248	✓	1	M.E.	
100-Min-Max	2	102	301	0.0291	0.409	✓	1	M.E.	
200-Min-Max	2	202	601	0.0803	0.643	✓	1	M.E.	
1-Range	2	4	5	0.00083	0.227	✓	1	T.O.	
10-Range	20	31	50	0.00797	0.611	✓	1	M.E.	
20-Range	40	61	100	0.0212	3.469	✓	1	M.E.	
40-Range	80	121	200	0.06242	35.89	✓	1	M.E.	
80-Range	160	241	400	0.25867	506.3	✓	1	M.E.	

Table 1: **Summary of experimental results for** DiPAut **and comparison with** CheckDP**. The columns in the table are as follows.** v **is the number of variables in the automaton.** s **is the number of states in the automaton.** $trans$ **is the number of transitions in the automaton. The weight calculation time and total time taken by** DiPAut **averaged over six executions are reported next, and are measured in seconds. Differentially private indicates if the automaton is differentially private or not. In case, it is not, we report the reason detected by the tool: DC/PV/LC/LP means that disclosing cycle/privacy-violating path/leaking cycle/leaking pair, respectively is detected.** $\mathfrak{D}$ **is the weight of the automaton computed by the algorithm in case it is differentially private. For** CheckDP**, the time column indicates the running time for CheckDP measured in seconds. The last column indicates the time taken for counterexample validation by** PSI **in case a counterexample is generated. T.O. denotes that the tool did not finish in** 30 **minutes. M.E. indicates that** CheckDP **reported a memory error.**

for any k as the outputs do not distinguish whether maximum or minimum is being updated in the first phase. However, it is well-formed and ϵ-differentially private. Psuedocode for k-Min-Max is shown as Algorithm 2.

Examples m-Range. Another set of examples for scalability is m-Range (for each m). m-Range is the m-dimensional version of Range. It repeatedly checks whether a sequence of points in the m-dimensional space is contained in a m-dimensional rectangle. The rectangle is specified by giving the upper and lower threshold for each coordinate of the rectangle. The algorithm initially adds Laplacian noise to each of these $2m$ thresholds, then processes the points by adding noise to each coordinate and checking that each noisy coordinate is within the noisy thresholds for that coordinate. Observe that m-Range is a set of examples, one for each m. For each m, the DiPA modeling m-Range has $2m$ variables, has $3m+1$ states and $5m$ transitions. For each m, m-Range satisfies output distinction, is well-formed, and is ϵ-differentially private. m-Range is given in Algorithm 3. Here the arrays T_1 and T_2 store the m-lower and m-upper thresholds, respectively. The arrays low and high store the noisy version of the lower and upper thresholds. In the experiments, T_1 is taken to be all 0s, and T_2 is taken to be all 1s.

Input: $q[1:m]$
Output: $out[1:Nm]$

for $j \leftarrow 1$ **to** m **do**
  low[j] $\leftarrow$ Lap($\frac{\epsilon}{4m}, T_1[j]$)
  high[j] $\leftarrow$ Lap($\frac{\epsilon}{4m}, T_2[j]$)
  $out[j] \leftarrow$ cont
end
for $i \leftarrow 1$ **to** N **do**
  for $j \leftarrow 1$ **to** m **do**
    r $\leftarrow$ Lap($\frac{\epsilon}{4}, q[m(i-1)+j]$)
    if (r $\geq$ low[j]) $\wedge$ (r $<$ high[j]) **then**
      $out[m(i-1)+j] \leftarrow$ cont
    else if ((r $\geq$ low[j]) $\wedge$ (r $>$ high[j])) **then**
      $out[m(i-1)+j] \leftarrow \top$
      exit
    end
    else if ((r $<$ low[j]) $\wedge$ (r $<$ high[j])) **then**
      $out[m(i-1)+j] \leftarrow \bot$
      exit
    end
  end
end

Algorithm 3: m-Range algorithm. m-Range is differentially private.

7.2 Summary of experimental results

The experimental results are summarized in Table 1. All experiments were run on a macOS computer with a 1.4 GHz Quad-Core Intel Core i5 CPU processor with 8GB RAM. The running time is benchmarked using pyperf [21], which runs each example 6 times and takes the average over the 6 instances. Figure 6 plots the running time of our implementation for k-Min-Max. As predicted, the tool confirms that k-Min-Max is ϵ-differentially private. A close examination of the algorithm for checking well-formedness reveals that the algorithm can check the well-formedness of k-Min-Max in time that is linear in k. This observation is confirmed by the experimental results. Note that the size of the DiPA modeling k-Min-Max is linear in k, and hence the running time is also linear in the size of DiPA. In contrast, a careful analysis reveals that the algorithm checking well-formedness takes time that is cubic in m for m-Range. This observation is also confirmed by the experimental results. (See Figure 7). As predicted, the tool confirms that m-Range

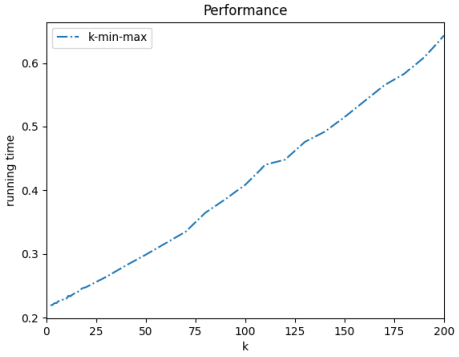

Figure 6: **Running time for k-Min-Max. The y-axis gives the running time measured in seconds, while the x-axis gives k. The size of the DiPA is linear in k. k-Min-Max is differentially private with weight 1.**

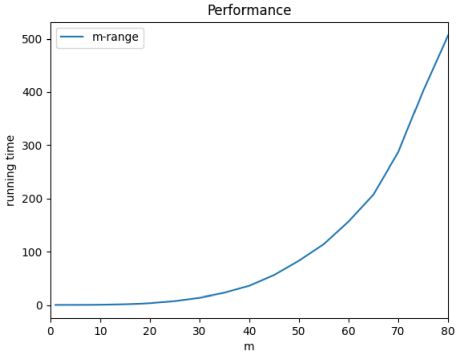

Figure 7: **Running time for m-Range. The y-axis gives the running time measured in seconds, while the x-axis gives m. The size of the DiPA is linear in m. m-Range is differentially private with weight 1.**

is ϵ-differentially private. Note that the number of variables in m-Range is $2m$, implying a quartic dependence on the number of variables as well. Data used to generate the graphs is given in [11].

Salient observations about our tool are as follows:

(1) DiPAut is able to check whether the algorithm described by a DiPA is well-formed in reasonable time.
(2) In case the automaton $\mathcal{A}$ is well-formed, it is able to compute a weight $\mathfrak{D}$ that $\mathcal{A}$ is $\mathfrak{D}\epsilon$-differentially private. The computed values match the theoretical values. Further, the computation of weight has little overhead.
(3) As predicted by the theory, the number of variables plays a crucial role in performance. While the theory predicts that this dependence is exponential (since the augmentation can be of exponential size), nevertheless, there are interesting examples in which the dependence is polynomial and not exponential.
(4) DiPAut is not only able to verify differential privacy for examples but also find violations of privacy in a reasonable time, as shown in Table 1.

Comparison with CheckDP. We compare the performance of our tool, DiPAut with CheckDP [29]. CheckDP employs the randomness alignment technique and attempts to prove differential privacy. If it fails to prove differential privacy, it generates a potential counterexample that must be validated using the PSI probabilistic model checker [24]. The key differences between CheckDP and DiPAut are as follows: (1) CheckDP supports other arithmetic operations besides comparison operators. (2) However, CheckDP is sound but incomplete and may fail to prove or disprove differential privacy. (3) CheckDP checks if a program is $\mathfrak{D}\epsilon$ differentially private for a given $\mathfrak{D}$. DiPAut, on the other hand, computes a $\mathfrak{D}$ for which the program is $\mathfrak{D}\epsilon$ differentially private. (4) DiPAut operates as a standalone tool, assessing the differential privacy of a given mechanism. The results of the comparison are summarized in Table 1. Apart from SVT and Num-Sparse, CheckDP times out on all other examples. For those two examples, DiPAut significantly outperforms CheckDP.

8 RELATED WORK

Online Programs and Comparison with [10]. The results in this paper are an extension of those presented in [10]. However, the automaton model proposed in [10] has only one storage variable, whereas we consider the generalization where the automaton has finitely many real-valued storage variables. Even though we use the same name for the automata model and for the conditions characterizing well-formed DiPA, the generalization to handle multiple real-valued storage variables is a significant extension. Defining leaking cycles, leaking pairs, privacy violating paths and disclosing cycles, requires a careful analysis of the ordering constraints imposed on values sampled in a run based on what gets stored in variables and the Boolean constraints that guard transitions. These concepts cannot be defined using just the underlying graph of the DiPA as in [10]; they require introducing the notion of a dependency graph of a run. Even with dependency graphs, the definition of these graph-theoretic conditions is subtle. For example, two cycles contained in a run may not form a leaking pair. However, they may become a leaking pair in an extension of the run as the additional transitions in the extension introduce new dependencies in the dependency graph (see Example 4 on Page 8). In the case of a single variable [10], such a situation does not arise.

Next, even though the proof showing that well-formedness is necessary for an output-distinct DiPA to be differentially private uses a strategy similar to the case for one variable [10], it is significantly more involved. For example, in showing that a leaking cycle is a witness to privacy violation, complications arise due to the need to track the dependency between multiple storage variables and the presence of non-input transitions. When constructing a pair of adjacent inputs that witness the violation of privacy, intervals of real numbers called *bands* need to be carefully identified, where the input of certain transitions is restricted to lie (see [11]). The proof that a leaking pair is a witness to privacy violation uses new ideas. In [10], the proof constructs, given $\mathfrak{D}$, two adjacent computations whose ratio is $> e^{\mathfrak{D}\epsilon}$ for each $\epsilon > 0$. In this paper, the adjacent computations have a ratio $> e^{\mathfrak{D}\epsilon}$ only for sufficiently large ϵ.

The proof showing that a well-formed DiPA is differentially private is also innovative. In [10], the proof is by induction on the

number of assignments to the stored variable in a run. In contrast, here the induction is on the number of transitions in a run, and the induction hypothesis is constructed by classifying the dependency graph nodes as gcycle_node or lcycle_node.

Privacy proof construction. Techniques based on type systems have been proposed in many papers [15, 16, 22, 27, 29, 31] for generating proofs of differential privacy. Some of these methods such as [15, 16, 22, 27] employ linear dependent types, for which the type-checking and type-inference may be challenging. In [1, 3–5] methods based on probabilistic couplings and random alignment arguments have been employed for proving differential privacy. Shadow execution-based method was introduced in [30]. Probabilistic I/O automata are used in [28] to model interactive differential privacy algorithms and simulation-based methods are used to verify differential privacy, but these methods have not been shown to be complete.

Counterexample generation. Automated techniques to search for privacy violations by generating counter examples have been proposed in [8, 17, 29]. Techniques include the use of statistical hypothesis testing [17], optimization techniques and symbolic differentiation [8] and program analysis [29]. These methods search over a bounded number of inputs.

Model-checking/Markov Chain approaches. Probabilistic model checking approach for verifying ϵ-differential privacy is employed in [12, 13, 25], where it is assumed that the program is given as a finite Markov Chain. These approaches do not allow for sampling from continuous random variables.

Decision Procedures. The decision problem of checking whether a randomized program is differentially private is studied in [2], where it is shown to be undecidable for programs with a single input and single output, assuming that the program can sample from Laplacian distributions. A decidable sub-class is identified where the inputs and outputs are constrained to be from a finite domain and have bounded length.

Complexity. Gaboardi et. al [23] study the complexity of deciding differential privacy for randomized Boolean circuits, and show that the problem is $\mathbf{coNP}^{\#\mathbf{P}}$-complete. The results are extended to Boolean programs [9] for which the verification problem is PSPACE-complete. In this line of work, programs have a finite number of inputs, the only probabilistic choices are fair coin tosses, and e^{ϵ} is taken to be a fixed rational number.

9 DISCUSSION

We discuss the restrictions used in various definitions in this paper.

Strong feasability. From the theoretical point of view, strong feasibility is used only to prove the necessity of well-formedness (Theorem 4). The sufficiency proof (Theorem 2) does not require the condition of strong feasibility. Nevertheless, we believe that all differential privacy mechanisms are strongly feasible. We have not encountered examples that violate the strong feasibility condition. Our intuition for this belief is as follows. First, any DiPA that *does not* have any non-input states is, by definition, strongly feasible. For DiPA with non-input states, the condition implies that the mean of the distribution at any two non-input states respects the order given by the dependency graph of a run. Let us consider the "deterministic" version of the automaton in which no noise is added. Intuitively, the "deterministic" version captures the behavior of the automaton in the limit as the privacy budget ϵ tends to infinity, i.e., becomes unlimited. A strongly feasible run implies that we can choose inputs such that the probability of that run tends to 1 as ϵ tends to ∞ and is executable in the "deterministic" version. A path that is not strongly feasible implies that the probability of this path tends to 0 as ϵ tends to ∞, irrespective of the choice of inputs, and will never be executed in the "deterministic version" because the insample values stored at the non-input states do not follow the order given by the dependency graph. The deterministic version of the automaton is relevant as a differentially private algorithm is often the noisy version of a deterministic algorithm (with noise added to make the automaton differentially private).

Output-distinction. Some examples do not meet the condition output distinction. For example, the k-Min-Max (See Section 7.1) and NoisyMax [20] are *not* output distinct. However, other examples (m-Range, SVT, NumericSparse) are output distinct. The output distinction condition is only needed to establish necessity but not for sufficiency. In other words, if an automaton is well-formed, it is differentially private, *even if it is not output distinct.* This is true for the k-Min-Max examples. However, the traditional NoisyMax is neither well-formed nor output distinct, and hence our technique does not establish its differential privacy. Some variants of NoisyMax (like checking if the kth input is maximum) are well-formed and hence can be handled by our techniques.

Adjaceny Relations. For algorithms working on a sequence of answers to queries on a database like SVT and Num-Sparse (see [20], pages 56 and 57), the assumption that queries are *1-sensitive* is common; here 1-sensitive means that adding or removing a member from a database can cause a difference of at most 1 in the output of each query. This assumption is satisfied by all counting queries and can be found in Algorithms 1, 2, 3 in [20] on pages 58, 62, 64, first paragraph on page 5 of [1] and third paragraph of Section 4 in [17].

More generally, our results also apply to a sequence of queries each of which is Δ-sensitive. The computation of $\mathfrak{D}$ will change, but the theorems of the sufficiency of well-formedness and necessity for well-formedness for output distinct DiPA remain true.

Boolean Guards on transitions in leaking cycle. In the definition of a leaking cycle (see Definition 8), it is possible that the constraint involving x in the guard of $\rho[i_2]$ is superfluous. When this happens, there have to be other variables in the guard of $\rho[i_2]$. However, we can show that after removing all superfluous checks from $\rho[i_2]$, either the original cycle will be a leaking cycle for some (possibly different) variable, or the leaking cycle gives rise to a leaking pair when repeated twice. Therefore, in principle, even a superfluous test does leak information (though indirectly).

The expressiveness of multi-variable DiPA vs one-variable DiPA. We can prove that multi-variable DiPA are strictly more expressive than one-variable DiPA. For example, we can formally show that the DiPA $\mathcal{A}_{\text{Range}}$ (See Figure 1) cannot be modeled using single-variable DiPA.

10 CONCLUSIONS

We extended the DiP automaton model introduced in [10] for modeling online algorithms that process a stream of unbounded real values representing answers to queries and, in response, produce a sequence of real or discrete output values. In the extended model, a DiPA $\mathcal{A}$ may use *multiple* storage variables to store noisy input values when executing transitions that are used in Boolean conditions that guard transitions. Our main contribution is a precise characterization of when DiPAs are differentially private using the notion of well-formed automata. The definition of well-formed automata is subtle and complicated, and requires the use of new graph structures associated with the runs of the automata, called dependency graphs. Well-formed DiPAs are shown to be differentially private and DiPAs satisfying the condition of output distinction that are differentially private are necessarily well-formed. The problem of checking well-formedness is PSPACE-complete. The algorithm for checking differential privacy has been implemented in a tool called DiPAut, and our experimental results demonstrate its promise.

As future work, it will be interesting to identify necessary conditions for classes of automata that do not satisfy the output distinction property. Extending DiPAs to allow a richer class of comparisons in the guards and a richer class of assignments, like using expressions involving additions of storage variables and/or constants in the guard conditions, is left for future exploration. Computing the optimal weight $\mathfrak{D}$ is another open problem.

Acknowledgements. The authors would like to thank Lipsy Gupta and the anonymous reviewers for their interesting and valuable comments. Rohit Chadha was partially supported by NSF CNS 1553548 and NSF CCF 1900924. A. Prasad Sistla was partially supported by NSF CCF 1901069, Mahesh Viswanathan was partially supported by NSF CCF 1901069 and NSF CCF 2007428, and Bishnu Bhusal was partially supported by NSF CCF 1900924.

REFERENCES

[1] Aws Albarghouti and Justin Hsu. 2018. Synthesizing coupling proofs of differential privacy. In *Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. 58:1–58:30.

[2] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2020. Deciding Differential Privacy for Programs with Finite Inputs and Outputs. In *35th Annual ACM/IEEE Symposium on Logic in Computer Science*. 141–154.

[3] Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Advanced Probabilistic Couplings for Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 55–67.

[4] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Proving differential privacy via probabilistic couplings. In *IEEE Symposium on Logic in Computer Science (LICS)*. 749–758.

[5] 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.

[6] David Beazley. 2022. GitHub - dabeaz/ply: Python Lex-Yacc — github.com. https://github.com/dabeaz/ply. [Accessed 24-Jan-2023].

[7] Bishnu Bhusal, Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. 2023. bhusalb/DiPAut: Version 1.0.1. https://doi.org/10.5281/zenodo.8332275

[8] 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)*. 508–524.

[9] Mark Bun, Marco Gaboardi, and Ludmila Glinskih. 2022. The Complexity of Verifying Boolean Programs as Differentially Private. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*. 396–411. https://doi.org/10.1109/CSF54842.2022.9919653

[10] Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. 2021. On Linear Time Decidability of Differential Privacy for Programs with Unbounded Inputs. In *36th Annual IEEE Symposium on Logic in Computer Science (LICS)*. 1–13.

[11] Rohit Chadha, A. Prasad Sistla, Mahesh Viswanathan, and Bishnu Bhusal. 2023. Deciding Differential Privacy of Online Algorithms with Multiple Variables. http://arxiv.org/

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

[13] Dmitry Chistikov, Stefan Kiefer, Andrzej S. Murawski, and David Purser. 2020. The Big-O Problem for Labelled Markov Chains and Weighted Automata. In *31st International Conference on Concurrency Theory (CONCUR) (LIPIcs)*, Vol. 171. 41:1–41:19.

[14] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. *InterJournal* Complex Systems (2006), 1695. https://igraph.org

[15] Arthur Azevedo de Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. 2014. Really Natural Linear Indexed Type Checking. In *26th 2014 International Symposium on Implementation and Application of Functional Languages (IFL)*. 5:1–5:12.

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

[17] Zeyu Ding, Yuxin Wang, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. 2018. Detecting Violations of Differential Privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 475–489.

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

[19] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan. 2009. On the complexity of differentially private data release: efficient algorithms and hardness results. In *ACM SIGACT Symposium on Theory of Computing (STOC)*. 381–390.

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

[21] Python Software Foundation. 2023. pyperf: A toolkit to write, run and analyze benchmarks. https://github.com/psf/pyperf. Accessed on 24-Jan-2023.

[22] 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)*. 357–370.

[23] Marco Gaboardi, Kobbi Nissim, and David Purser. 2020. The Complexity of Verifying Loop-Free Programs as Differentially Private. In *47th International Colloquium on Automata, Languages, and Programming, (ICALP) (LIPIcs)*, Vol. 168. 129:1–129:17.

[24] Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *International Conference on Computer Aided Verification*. Springer, 62–83.

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

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

[27] 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 (ICFP)*. 157–168.

[28] Michael Carl Tschantz, Dilsun Kirli Kaynar, and Anupam Datta. 2011. Formal Verification of Differential Privacy for Interactive Systems (Extended Abstract). In *27th Conference on the Mathematical Foundations of Programming Semantics (MFPS) (Electronic Notes in Theoretical Computer Science)*, Vol. 276. 61–79.

[29] Yuxin Wang, Zeyu Ding, Daniel Kifer, and Danfeng Zhang. 2020. CheckDP: An Automated and Integrated Approach for Proving Differential Privacy or Finding Precise Counterexamples. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 919–938.

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

[31] 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)*. 888–901.