1 Introduction

Differential privacy is a framework for privacy that gives rigorous guarantees on the amount of data leakage any one person's data can be subjected to when releasing statistical data. Since being introduced in 2006 [7], differential privacy has become the gold standard for private statistical analysis. Differentially private algorithms, whose efficacy are characterized by a "privacy cost" ε, primarily rely on the addition of statistical noise, ensuring that statistical results remain approximately correct while preventing any one person's information from being revealed.

Differentially private algorithms are notoriously tricky to analyze for correctness; most famously, the Sparse Vector Technique (SVT) algorithm has gone through multiple iterations, some of which were later shown to completely fail at protecting privacy[10]. Previous implementations of differential privacy by Apple have similarly been shown to have an increase from the claimed privacy cost by a factor of up to 16 [11].

Thus, much work has been done on developing methods for automatic verification of differentially private algorithms, both in their overall privacy and in the specific privacy costs they claim to achieve. Because even for limited programs the problem of determining if a program is differentially private is undecidable[2], previous work tends to focus on semi-decidability or further restricting program models.

Recently, a line of work has emerged around **approximate liftings** [3, 5, 4, 9]. Approximate liftings are a generalization of probabilistic couplings, themselves a well-known technique in probability theory for analyzing relationships between random variables. Approximate liftings allow for a more structured proof approach to many algorithms that themselves are not conducive to a standard compositional analysis, such as SVT. Because of their structure, liftings also lend themselves to automated proof construction [1].

We first rewrite the major results of approximate liftings in {not program logic}[1]. We then use approximate liftings to demonstrate that a certain limited class of programs, first described in [6], are differentially private; interestingly, we show that our class of liftings completely characterizes this class of programs. Additionally, we demonstrate that the privacy of a natural generalization of this class of programs can be proven using liftings and almost immediately follows from the privacy of the smaller class.

2 Differential Privacy

Differential privacy is a mathematically robust approach to privacy; most generally, differential privacy ensures that it is unlikely for an adversary to distinguish between whether or not one person's data was used in a private algorithm. To do this, differentially private algorithms rely on randomization, especially through the addition of statistical noise.

More precisely then, for a fixed output σ of a private algorithm A, the probability of obtaining σ for a dataset with some individual Alex is close (measured by a multiplicative factor) to the

[1]not sure how to describe this, also not sure if worth mentioning

probability of obtaining σ for the same dataset with Alex removed or Alex's data changed.

We will consider **datasets** $\mathcal{X} \in X^n$ of size n, where X is the set of all possible rows in the dataset; each person is represented by a single row.

We next define what it means for datasets to be "similar" to each other.

Definition 2.1. Two datasets $\mathcal{X} = (x_1, \dots, x_n), \mathcal{X}' = (x'_1, \dots, x'_n) \in X^n$ are **adjacent** (notated $\mathcal{X} \sim \mathcal{X}'$) if $|\{i : x_i \neq x'_i\}| \leq 1$[2].

We thus formalize privacy under this framework as follows.

Definition 2.2 (Pure Differential Privacy). A randomized algorithm A is ε-differentially private if, for all pairs of **adjacent** datasets X and X' and all events $E \subseteq \texttt{im}(A)$,

$$\mathbb{P}[A(X) \in E] \leq e^{\varepsilon}\mathbb{P}[A(X') \in E]$$

An extremely useful property of differential privacy is that differentially private programs can be **sequentially composed** with a linear degradation in privacy:

Theorem 2.3 (Standard Composition). *If A is ε_1-differentially private and, for all σ, $B(\sigma, \cdot)$ is ε_2-differentially private, then $B(A(X), X)$ is $\varepsilon_1 + \varepsilon_2$-differentially private.*

Composition therefore allows us to view privacy parameters ε as a "budget" for privacy-leaking operations in a program. Many[3] common differentially private algorithms are thus built out of well-known private components combined together, which also lend themselves to straightforward analyses.

2.1 Sensitivity and the Laplace Mechanism

Because we are typically interested in analyzing *functions* of our raw dataset (for example, the average age of a town), it is often useful to examine differential privacy through a similar model - instead of comparing two adjacent datasets $X \sim X'$, we compare **queries** $f(X)$ and $f(X')$. In this world, we care about the *sensitivity* of functions: how much a function *changes* when considering adjacent inputs.

Definition 2.4. The (ℓ_1-)sensitivity of a function $f : X \to \mathbb{R}$, often denoted Δf, is defined as $\Delta f = \max_{X \sim X'} ||f(X) - f(X')||_1$.

Given a function's sensitivity, we can easily make it differentially private through the use of the **Laplace Mechanism**.

Definition 2.5. The Laplace distribution $\texttt{Lap}(\mu, b)$ with mean μ and spread parameter b is the probability distribution with probability density function $f(x) = \frac{1}{2b}\exp(-\frac{|x-\mu|}{b})$. If $\mu = 0$, we will often abbreviate $\texttt{Lap}(0, b)$ as $\texttt{Lap}(b)$.

[2] A common variant is to define adjacency by the removal or addition of an entry, rather than changing one

[3] generic platitude - reword

The Laplace Mechanism, as expected, simply adds noise sampled from the Laplace distribution to a query result.

Theorem 2.6 (Theorem 3.6 [8])**.** *For a function f with sensitivity Δ, $A(X) = f(X) + \texttt{Lap}(\frac{\Delta}{\varepsilon})$ is ε-differentially private.*

We will consider the scenario where we are given a potentially infinite *sequence* of real-valued query functions $q_0, q_1, \ldots$, each with sensitivity at most Δ.

2.2 Deciding Privacy

Because designing differentially private algorithms can be quite tricky, we would like to be able to automatically (i.e. algorithmically) verify whether or not a given program is private, especially for algorithms whose privacy proofs do not rely primarily on composition. Ideally, beyond just determining whether a program is private or not, if a program is private, we'd like to find a good bound on the privacy cost for the program as well.

Unfortunately, even for relatively simple programs, just the basic problem is undecidable.

Theorem 2.7 ([2])**.** *The problem of determining whether a program from a certain class of algorithms with assignments, conditionals, and while loops is ε-differentially private is undecidable*[4]*.*

Thus, we will derive a decision procedure for a very specific class of potentially private programs; in particular, this class of programs lends itself to a straightforward analysis by **approximate liftings**, which we now introduce.

3 Couplings and Liftings

Probabilistic couplings are a common tool in probability theory; intuitively, couplings allow for the joint analysis of nominally unrelated probabilistic processes.

Definition 3.1. A coupling between two distributions A and B is a joint distribution C such that $\pi_1(C) = A$ and $\pi_2(C) = B$, where $\pi_1(C)$ and $\pi_2(C)$ are the first and second marginals of C, respectively.

In particular, couplings can be useful when analyzing the relation between two probablistic processes; couplings were first formulated by [check name] to analyze the behaviour of markov chains and have close connections to concepts such as total variation distance and stochastic domination.

As useful as standard couplings are, however, we must use more powerful machinery to properly reason about privacy.

Approximate liftings [4, 5, 9, 3] allow us to apply couplings to the realm of differential privacy.

[4]rephrase?

Definition 3.2. Let A_1, A_2 be two probability spaces[5]. We say a distribution μ_1 on A_1 and μ_2 on A_2 are related by the **ε-lifting** of the relation $\Psi \subseteq A_1 \times A_2$ (written $\mu_1 \Psi^{\#\varepsilon} \mu_2$) if there exist two **witness distributions** μ_L, μ_R on $A_1 \times A_2$ such that

1. $\pi_1(\mu_L) = \mu_1$ and $\pi_2(\mu_R) = \mu_2$
2. $\texttt{supp}(\mu_L), \texttt{supp}(\mu_R) \subseteq \Psi$
3. $\sup_{E \subseteq A_1 \times A_2} (\mathbb{P}_{x \leftarrow \mu_L}[x \in E] - e^{\varepsilon} \mathbb{P}_{x \leftarrow \mu_R}[x \in E]) \leq 0$

The similarities between the third condition and the definition of differential privacy should be clear. Indeed, there is a close connection between approximate liftings and differential privacy:

Theorem 3.3. *An algorithm $A(X)$ is ε-differentially private if and only if, for all adjacent input sequences $X \sim X'$, $A(X)(=)^{\#\varepsilon} A(X')$.*

If we are solely aiming to show that a program is private, we can instead work with the following relaxation:

Theorem 3.4. *If for all adjacent input sequences $X \sim X'$, $A(X)\{(a,b) : a = \sigma \implies b = \sigma\}^{\#\varepsilon} A(X')$, then $A(X)$ is $\varepsilon-$differentially private.*

As expected, the foundational results of differential privacy can be restated in terms of liftings:

Proposition 3.5 (Laplace Mechanism for Liftings)**.** *If $X_1 \sim \texttt{Lap}(\mu_1, \frac{1}{\varepsilon})$ and $X_2 \sim \texttt{Lap}(\mu_2, \frac{1}{\varepsilon})$, then $X_1(=)^{\#\varepsilon|\mu_1 - \mu_2|} X_2$.*

Theorem 3.6 (Composition of Liftings)**.** *Let A_1, B_2, A_2, B_2 be distributions over S_1, T_1, S_2, T_2, respectively and let $R_1 \subseteq S_1 \times T_1$, $R_2 \subseteq S_2 \times T_2$ be relations. If $A_1 R_1^{\#\varepsilon_1} B_1$ and $A_1 R_1 B_1 \implies A_2 R_2^{\#\varepsilon_2} B_2$, then $A_2 R_2^{\#\varepsilon_1 + \varepsilon_2} B_2$.*

The structure of theorems 3.4 and 3.6 suggests the format that coupling proofs of privacy take: given two "runs" of an algorithm on adjacent inputs, construct many smaller liftings between program variables in each run and compose these liftings together to show that a final implicatory lifting between the outputs of the two runs exists.

3.1 Proving SVT with couplings

A classic algorithm that requires analysis beyond standard composition is Sparse Vector Technique (SVT). Given a possibly infinite stream of inputs and a threshold value, SVT will output if the queries are above or below the threshold (with noise on both the query and the threshold).

Unusually for differentially private algorithms, SVT can output a potentially unbounded number of "below threshold" queries before the first c "above threshold"s (or vice-versa), where c is some constant set by the user; when $c = 1$, SVT is frequently also referred to as "Above (or Below) Threshold". Potential applications include, for example, checking that

[5]may need to formally rewrite this at some point

a series of inputs is within an expected range or, appropriately, privately determining the non-zero elements of a sparse vector.

Because SVT allows for a potentially unbounded number of "below threshold" query outputs, its analysis requires a non-standard approach; a naive composition approach that assigns a fixed cost to outputting the result of each query will immediately result in unbounded privacy cost as well. Indeed, the analysis of SVT is notoriously difficult, with multiple published attempts at privacy proofs that were later shown to be incorrect[6].

However, re-analyzing SVT using approximate liftings can be relatively simple.

Algorithm 1 Sparse Vector Technique

```
    Input: $\mathcal{X} \in X^n$, $T \in \mathbb{R}$, $Q = q_1, \ldots \in (X^n \to \mathbb{R})^*$ with sensitivity $\Delta$, $c \in \mathbb{N}$.
 1: $\varepsilon_1, \varepsilon_2 \leftarrow \frac{\varepsilon}{2}, \rho \leftarrow \mathtt{Lap}(\frac{\Delta}{\varepsilon_1})$, $count \leftarrow 0$
 2: for $q_i \in Q$ do
 3:     $z \leftarrow \mathtt{Lap}(\frac{2c\Delta}{\varepsilon_2})$
 4:     if $q_i(\mathcal{X}) + z \geq T + \rho$ then
 5:         output $\top$
 6:         $count \leftarrow count + 1$
 7:         if $count \geq c$ then
 8:             break
 9:         end if
10:     else
11:         output $\bot$
12:     end if
13: end for
```

Theorem 3.7. *Sparse Vector Technique is ε-differentially private.*

Proof. Consider two runs of SVT with adjacent inputs $\mathcal{X} \sim \mathcal{X}'$, respectively. We are aiming to show that $SVT(\mathcal{X}, T, Q, c)\{(a, b) : a = \sigma \implies b = \sigma\}^{\#\varepsilon} SVT(\mathcal{X}', T, Q, c)$ is a valid lifting.

Fix some output $\sigma \in \{\bot, \top\}^n$. Let $A = \{i : \sigma_i = \top\}$ be the indices of queries that are measured to be above the threshold. Note that $|A| = c$.

For every program variable x, let $x\langle 1\rangle$ and $x\langle 2\rangle$ represent the value of x in $SVT(\mathcal{X}, T, Q, c)$ and $SVT(\mathcal{X}', T, Q, c)$, respectively, so, for example, $q_i(\mathcal{X})\langle 1\rangle = q_i(\mathcal{X})$ and $q_i(\mathcal{X})\langle 2\rangle = q_i(\mathcal{X}')$.

Let $\tilde{T} = T + \rho$. Then $\tilde{T} \sim \mathtt{Lap}(T, \frac{\Delta}{\varepsilon_1})$, so $\tilde{T}\langle 1\rangle + \Delta (=)^{\#\varepsilon_1} \tilde{T}\langle 2\rangle$.

Let $S_i = q_i(\mathcal{X}) + z_i$, so $S_i \sim \mathtt{Lap}(q_i(\mathcal{X}), \frac{2c\Delta}{\varepsilon_2})$.

For all i such that $0 \leq i < n$, $i \notin A$, we construct the lifting $z_i\langle 1\rangle (=)^{\#0} z_i\langle 2\rangle$.

Then note that $\tilde{T}\langle 1\rangle + \Delta = \tilde{T}\langle 2\rangle \wedge z_i\langle 1\rangle = z_i\langle 2\rangle \implies (S_i\langle 1\rangle < \tilde{T}\langle 1\rangle \implies S_i\langle 2\rangle < \tilde{T}\langle 2\rangle)$.

[6]A textbook analysis of SVT, along with a discussion of bugged versions and incorrect privacy proofs, can be found at [10]

For all $i \in A$, create the lifting $z_i\langle 1\rangle (=)^{\#\frac{\varepsilon_2}{c}} z_i\langle 2\rangle - q_i(\mathcal{X}) + q_i(\mathcal{X}') - \Delta$, or equivalently, $S_i\langle 1\rangle + \Delta (=)^{\#\frac{\varepsilon_2}{c}} S_i\langle 2\rangle$. Note that this costs $\frac{\varepsilon_2}{c}$ since $|q_i(\mathcal{X}) - q_i(\mathcal{X}')| \leq \Delta$.

Then

$$\tilde{T}\langle 1\rangle + \Delta = \tilde{T}\langle 2\rangle \wedge S_i\langle 1\rangle + \Delta = S_i\langle 2\rangle \implies (S_i\langle 1\rangle \geq \tilde{T}\langle 1\rangle \implies S_i\langle 2\rangle \geq \tilde{T}\langle 2\rangle)$$

Thus, for all i, $SVT(\mathcal{X}, T, Q, c)_i = \sigma_i \implies SVT(\mathcal{X}', T, Q, c)_i = \sigma_i$, so $SVT(\mathcal{X}, T, Q, c)\{(a, b) : a = \sigma \implies b = \sigma\}^{\#\varepsilon_1+\varepsilon_2} SVT(\mathcal{X}', T, Q, c)$.

By Theorem 3.4, SVT is ε-differentially private. □

4 Segments

We begin by building up a program model for SVT-style algorithms. There are three major components of SVT: taking in a threshold value and adding Laplace noise to it, taking in input and adding Laplace noise to it, and comparing the noisy threshold to the noisy input.

4.1 Individual Transitions

We will model programs as finite state automata, with each state representing a Under this paradigm, we begin with the simplest possible program: a single transition between program states.

At each program state, we will store one persistent real-valued variable called `x`. Additionally, each state q has real-valued Laplace noise parameters (d_q, d'_q) associated with it.

At each state, a real valued input `in` is read in and Laplace noise sampled from the distribution $\texttt{Lap}(0, \frac{1}{d\varepsilon})$ is added to it and stored in a temporary variable `insample`. Another temporary variable $\texttt{insample}'$ stores the sum of Laplace noise sampled from the distribution $\texttt{Lap}(0, \frac{1}{d'\varepsilon})$ and `in`.

Fundamentally, our program will transition between states based on a comparison between `insample` and `x` at each state.

Then, given two states q, q' and a transition t between them, a program can move from state q to q' if

4.2 A Program Model

We start with a basic, "straight-line" program model.

Definition 4.1. A simple segment automaton (SSA) $A(x)$ with initial value $x \in \mathbb{R}$ is a tuple $(Q, \Sigma, C, \Gamma, q_{init}, X, P, \delta)$ where

- Q is a finite set of states
- $\Sigma = \mathbb{R}$ is the input alphabet

- $C = \{\texttt{true}, \texttt{insample} < x, \texttt{insample} \geq x\}$ is a set of transition **guards**
- Γ is a finite output alphabet
- $q_{init} \in Q$ is the starting state
- $X = \{\texttt{x}, \texttt{insample}, \texttt{insample}'\}$ is a set of real-valued program variables
- $P : Q \to \mathbb{Q} \times \mathbb{Q}^{\geq 0} \times \mathbb{Q} \times \mathbb{Q}^{\geq 0}$ is a function that assigns a series of Laplace distribution parameters to each state for sampling
- $\delta : (Q \times C) \to Q \times (\Gamma \cup \{\texttt{insample}, \texttt{insample}'\})$ is a (partial) transition function.

Additionally, δ is restricted such that

- For every state $q \in Q$, $\delta(q, c)$ can be defined for **at most one** $c \in C$.
- There are no cycles in the graph of A; i.e., for every path $\rho = q_{init} \to q_1 \to \ldots \to q_n$ in A starting from q_{init}, q_{init} and q_i are all pairwise distinct.

Note in particular that $A(x)$ is parameterized by some initial real value x, stored in the variable $\texttt{x}$. Importantly, we will treat x as a random variable drawn from some Laplace distribution.

An SSA $A(x)$ runs as follows:

- The variable $\texttt{x}$ has a stored real value, which is initialized to x.
- At **every** state $q \in Q$, A will read in some real-valued input value a. Here, a corresponds to the "true" value of a query function $q_i(X)$. Let $P(q) = (\mu, b, \mu', b')$. The variables $\texttt{insample}$ and $\texttt{insample}'$ are initialized such that $\texttt{insample} = a + \texttt{Lap}(\mu, b)$ and $\texttt{insample}' = a + \texttt{Lap}(\mu', b')$.
- If $\delta(q, \texttt{true}) = (q', \sigma)$, then A transitions to q' and outputs $\sigma \in (\Gamma \cup \{\texttt{insample}, \texttt{insample}'\})$. Similarly if $\texttt{insample} < x$ (symmetrically, $\texttt{insample} \geq x$) and $\delta(q, \texttt{insample} < x) = (q', \sigma)$ (symmetrically, $\delta(q, \texttt{insample} \geq x) = (q', \sigma)$), then then A transitions to q' and outputs $\sigma \in (\Gamma \cup \{\texttt{insample}, \texttt{insample}'\})$.
- If, at a state $q \in Q$, a guard c is valid (e.g. $c = \texttt{insample} < x$ and $\texttt{insample} < x$) and $\delta(q, c)$ is not defined (including if $\delta(q, c)$ is not defined for any c), the automaton terminates.
- If $q = q_{init}$, then $\texttt{x}$ is **assigned** so that $\texttt{x} \leftarrow \texttt{insample}$ after the transition comparison.

Note that, without loss of generality, we can assume that $P(q) = (0, b, 0, b')$ by shifting the inputs appropriately. Thus, we will frequently abuse notation and write $P(q) = (b, b')$.

Definition 4.2. A path ρ of an SSA $A(x)$ is a sequence of states $r_0 \to r_1 \to \ldots \to r_n$ such that for every $i \in \{0, 1, \ldots, n-1\}$, there exists some guard $c \in C$ such that $\delta(q_i, c) = (q_{i+1}, \sigma)$ for some output σ.

Given a path $\rho = r_0 \to r_1 \to \ldots \to r_n$ in A, we will notate the transition $r_i \to r_{i+1}$ with $\texttt{trans}(r_i)$ and the guard (i.e. $\texttt{true}, \texttt{insample} < x$, or $\texttt{insample} \geq x$) of $\texttt{trans}(r_i)$ as $\texttt{guard}(r_i)$.

For technical reasons, whenever either $\texttt{insample}$ or $\texttt{insample}'$ are output by an SSA, we must define an interval (a, b) that contains the output. Thus, when a transition outputs $\texttt{insample}$ or $\texttt{insample}'$, we say that σ is of the form $(\texttt{insample}, c, d)$ or $(\texttt{insample}', c, d)$ for $c, d \in R_\infty$.

If such a tuple is output, it can be interpreted as $\texttt{insample}$ or $\texttt{insample}'$ being output with the outputted real value falling within the interval (c, d).

Importantly, this allows us to define the **probability** of an SSA traversing a specified path with a specified output.

Definition 4.3. Let $A(x)$ be an SSA, $\rho = r_0 \to r_1 \to \ldots \to r_n$ be a path in A, and let σ be a possible output of A. Additionally, let $\alpha \in \mathbb{R}^*$ be a sequence of real-valued inputs.

Then $\mathbb{P}[x, \rho^{(\alpha,\sigma)}]$ is the probability of A outputting σ while traversing path ρ with $\texttt{x} = x$ at the initial state and input sequence α.

In particular, we can define $\mathbb{P}[x, \rho^{(\alpha,\sigma)}]$ inductively. Let $\delta(r_0, \texttt{guard}(r_0)) = (q, \sigma_0)$ and $P(r_0) = (\mu, b, \mu', b')$. If $\sigma_0 \in \{\texttt{insample}, \texttt{insample}'\}$, then let $\sigma_0 = (\texttt{insample}, c, d)$ or $\sigma_0 = (\texttt{insample}', c', d')$ as appropriate.

Additionally, let $(u, v), (u', v') \in \mathbb{R}_\infty$ be defined as follows:

$$(u, v) = \begin{cases} (-\infty, \infty) & \texttt{guard}(r_0) = \texttt{true} \wedge \sigma_0 \neq \texttt{insample} \\ (c, d) & \texttt{guard}(r_0) = \texttt{true} \wedge \sigma_0 = \texttt{insample} \\ (-\infty, x) & \texttt{guard}(r_0) = \texttt{insample} < x \wedge \sigma_0 \neq \texttt{insample} \\ (c, \min(x, d)) & \texttt{guard}(r_0) = \texttt{insample} < x \wedge \sigma_0 = \texttt{insample} \\ (x, \infty) & \texttt{guard}(r_0) = \texttt{insample} \geq x \wedge \sigma_0 \neq \texttt{insample} \\ (\max(x, c), d) & \texttt{guard}(r_0) = \texttt{insample} \geq x \wedge \sigma_0 = \texttt{insample} \end{cases}$$

$$(u, v) = (c', d')$$

Then

$$\mathbb{P}[x, \rho^{(\alpha,\sigma)}] = \begin{cases} 1 & |\rho| = 0 \\ \int_u^v \texttt{Lap}_{\mu+a,b}(z) dz \mathbb{P}[x, tail(\rho^{(\alpha,\sigma)})] & \sigma_0 \neq \texttt{insample}' \wedge r_0 \neq q_{init} \\ \int_{u'}^{v'} \texttt{Lap}_{\mu'+a,b'}(z') dz' \int_u^v \texttt{Lap}_{\mu+a,b}(z) dz \mathbb{P}[x, tail(\rho^{(\alpha,\sigma)})] & \sigma_0 = \texttt{insample}' \wedge r_0 \neq q_{init} \\ \int_u^v \texttt{Lap}_{\mu+a,b}(z) \mathbb{P}[z, tail(\rho^{(\alpha,\sigma)})] dz & \sigma_0 \neq \texttt{insample}' \wedge r_0 = q_{init} \\ \int_{u'}^{v'} \texttt{Lap}_{\mu'+a,b'}(z') dz' \int_u^v \texttt{Lap}_{\mu+a,b}(z) \mathbb{P}[z, tail(\rho^{(\alpha,\sigma)})] dz & \sigma_0 = \texttt{insample}' \wedge r_0 = q_{init} \end{cases}$$

where $\texttt{Lap}_{\mu,b}(x)$ is the PDF of a Laplace distribution with mean μ and spread parameter b and $tail(\rho) = r_1 \to \ldots \to r_n$.

We now work towards defining what it means for an SSA to be differentially private. Recall that α is, in reality, a sequence of **functions** of some underlying dataset. This means that adjacency in this context is defined as follows:

Definition 4.4. Two input sequences α, α' of length n are Δ-adjacent if, for all $i \in [n]$, $|\alpha_i - \alpha'_i| \leq \Delta$.

Similarly, two paths $\rho^{(\alpha,\sigma)}$ and $\rho^{(\alpha',\sigma)}$ are Δ-adjacent if α and α' are adjacent.

If Δ is not specified, we assume that $\Delta = 1$.

We can now define what it means for SSAs to be **differentially private**.

Definition 4.5. An SSA $A(x)$ with initial value x is $d\varepsilon$-differentially private if there exists some $d > 0$ such that $\forall \varepsilon > 0$, for all paths ρ, possible outputs σ, and adjacent input sequences $\alpha \sim \alpha'$, $\mathbb{P}[x, \rho^{(\alpha,\sigma)}] \leq e^{d\varepsilon}\mathbb{P}[x, \rho^{(\alpha',\sigma)}]$.

Recall that x is treated as a random variable drawn from a Laplace distribution.

Note that we slightly redefine ε-differential privacy as $d\varepsilon$-differential privacy, treating ε as a universal scaling parameter that can be fine-tuned by users for their own purposes. In particular, we argue that this definition is functionally equivalent[7], since if we target ε^*-differential privacy, we can always take $\varepsilon = \frac{\varepsilon^*}{d}$.

4.3 Coupling Strategies

For an SSA $A(x)$ as defined above, there are a finite set of **strategies**, using approximate liftings, that we can use to show that $A(x)$ is $d\varepsilon$-differentially private.

These **coupling strategies** are comprised of a series of liftings that, when combined together, will prove that A is private.

As a preliminary step, we fix $\varepsilon > 0$ as a generic privacy scale parameter, as described previously.

4.3.1 A Discrete Set of Coupling Strategies

First, fix some possible output σ of A. Note that this implicitly also determines some path ρ in A, since SSAs have no branching. Fix two adjacent datasets $X \sim X'$; this also determines two adjacent input sequences α, α'. We will consider the adjacent paths $\rho^{(\alpha,\sigma)}$ and $\rho^{(\alpha',\sigma)}$

For this path, we are aiming to construct the lifting $\rho(X)\{(a, b) : a = o \implies b = o\}^{\#d\varepsilon}\rho(X')$.

In particular, because SSAs have no branching, we must only ensure that all transitions in ρ are taken and, if a real-value r is output, that $A(X)$ outputting r implies $A(X')$ outputting r as well.

We claim that, in all cases, we can select one of three **coupling strategies** that will allow us to construct the larger lifting for the entire segment.

We will first set up some preliminary notation. Let $\rho = r_0 \to r_1 \to \ldots r_n$ where $r_0 = q_{init}$ and let $\sigma = \sigma_1\sigma_2 \cdots \sigma_n$.

[7][6] notes that it is not entirely clear how this differs from standard differential privacy, but that the known decidability result does not apply here - maybe something to investigate

For all states r_i, let $P(r_i) = (\frac{1}{d_i\varepsilon}, \frac{1}{d'_i\varepsilon})$ and let $z_i \sim \texttt{Lap}(\frac{1}{d_i\varepsilon}), z'_i \sim \texttt{Lap}(\frac{1}{d'_i\varepsilon})$ be the noises added to the input for `insample` and `insample`$'$ at each state, respectively. Additionally, suppose that $\texttt{x} = x_0 \sim \texttt{Lap}(\mu_x, \frac{1}{d_x\varepsilon})$ initially.

Similarly, let $\texttt{in}_i$ for $i \in \{0, \dots, n\}$ represent the input value read at each state r_i and let x be the value of the stored variable of A after q_0. Note that $\texttt{insample}_i = \texttt{in}_i + z_i$ by definition.

For all of these variables v, let $v\langle 1\rangle$ represent its value in a run of $A(X)$, and let $v\langle 2\rangle$ represent its value in a run of $A(X')$. For example, this means that $\texttt{in}_i\langle 1\rangle = \alpha_i$ and $\texttt{in}_i\langle 2\rangle = \alpha'_i$. In particular, we will overload notation and use $\sigma\langle k\rangle$ to represent the random variable corresponding to the output of run k[8].

We call the three coupling strategies we can apply S^L, S^G, and S^N.

Coupling Strategies 1 and 2: S^L and S^G.

Suppose that $\texttt{trans}(r_0)$ does not output `insample`. If it does, then we cannot use S^L, and must use S^N.

Create the lifting $z_0\langle 1\rangle (=)^{\#2d_0\varepsilon} z_0\langle 2\rangle + \texttt{in}_0\langle 2\rangle - \texttt{in}_0\langle 1\rangle - 1$. Note that since $\texttt{insample}_0$ gets assigned into `x` and $\texttt{insample}_0 = \texttt{in}_0 + z_0$, this is equivalent to constructing the lifting $x\langle 1\rangle + 1(=)^{\#2d_0\varepsilon} x\langle 2\rangle$.

Recall that $|\texttt{in}_0\langle 1\rangle - \texttt{in}_0\langle 2\rangle| \leq 1$ since the input sequences α, α' are (1-)adjacent; this is why the lifting has a cost of $2d_0\varepsilon$.

We must ensure that the initial transition guard $\texttt{guard}(r_0)$ is satisfied. This is a unique situation because this guard is dependent on the *initial* value of $\texttt{x} = x_0$; all later guards will depend on the value that `x` takes *after* this transition.

If $\texttt{guard}(r_0) = \texttt{true}$ this is trivially satisfied. Otherwise, if $\texttt{guard}(r_0) = \texttt{insample} < x$, then if we construct $x_0\langle 1\rangle + 1(=)^{\#d_x\varepsilon} x_0\langle 2\rangle$, we have that $\texttt{insample}_0\langle 1\rangle < x_0\langle 1\rangle + 1 \implies \texttt{insample}_0\langle 2\rangle - 1 < x_0\langle 2\rangle - 1$.

Similarly, if $\texttt{guard}(r_0) = \texttt{insample} \geq x$, then if we construct $x_0\langle 1\rangle(=)^{\#0} x_0\langle 2\rangle$, we have that $\texttt{insample}_0\langle 1\rangle \geq x_0\langle 1\rangle \implies \texttt{insample}_0\langle 2\rangle \geq x_0\langle 2\rangle$.

Although we can freely couple $x_0\langle 1\rangle$ and $x_0\langle 2\rangle$ together in this case, we will see that with more complicated families of automata, there may be additional restrictions on x_0 that constrain our ability to do so in the future.

We now consider all transitions after the initial transition. Fix some $i \in \{1, \dots, n-1\}$.

If $\texttt{trans}(r_i)$ outputs `insample`, then construct the lifting $z_i\langle 1\rangle(=)^{\#d_i} z_i\langle 2\rangle + \texttt{in}_i\langle 2\rangle - \texttt{in}_0\langle 1\rangle$. This is equivalent to constructing the lifting $\texttt{insample}\langle 1\rangle(=)^{\#d_i}\texttt{insample}\langle 2\rangle$.

Similarly, if $\texttt{trans}(r_i)$ outputs `insample`$'$, then construct the lifting
$\texttt{insample}'\langle 1\rangle(=)^{\#(d'_i,0)}\texttt{insample}'\langle 2\rangle$.

[8]As distinguished from the fixed output σ from before

Otherwise if $\texttt{guard}(r_i) = \texttt{insample} < x$, construct the lifting $z_i\langle 1\rangle (=)^{\#(0,0)} z_i\langle 2\rangle$. If $\texttt{trans}(r_i)$ doesn't output $\texttt{insample}$ and $\texttt{guard}(r_i) = \texttt{insample} \geq x$, construct the lifting $z_i\langle 1\rangle + 2(=)^{\#(2\epsilon_i,0)} z_i\langle 2\rangle$.

Because every transition must have an output, we can construct our final lifting by creating liftings for each character of the output; i.e. creating the liftings $\sigma_i\langle 1\rangle\{(a,b) : o = a \implies o = b\}^{\#d_i}\sigma_i\langle 2\rangle$ for all i.

We claim that if $\sigma_i \in \Gamma$ and $\texttt{guard}(a_i) \in \{\texttt{insample} < x, \texttt{true}\}$, we can construct the lifting $\sigma_i\langle 1\rangle\{(a,b) : a = \sigma_i \implies b = \sigma_i\}^{\#0}\sigma_i\langle 2\rangle$.

We demonstrate why this holds for $\texttt{guard}(a_i) = \texttt{insample} < x$:

$$\begin{aligned}
\sigma_i\langle 1\rangle = \sigma_i &\implies A(X) \text{ takes } \texttt{trans}(a_i)\\
&\implies \texttt{in}_i\langle 1\rangle + z_i\langle 1\rangle < x\langle 1\rangle\\
&\implies \texttt{in}_i\langle 1\rangle + 1 + z_i\langle 2\rangle < x\langle 2\rangle\\
&\implies \texttt{in}_i\langle 2\rangle + z_i\langle 2\rangle < x\langle 2\rangle\\
&\implies \mathcal{A}(X') \text{ takes } \texttt{trans}(a_i)\\
&\implies \sigma_i\langle 2\rangle = \sigma_i
\end{aligned}$$

When $\sigma_i = \texttt{insample}$ or $\sigma_i = \texttt{insample}'$, a similar analysis holds, with the additional caveat that $\texttt{insample}\langle 1\rangle = \texttt{insample}\langle 2\rangle$ and $\texttt{insample}'\langle 1\rangle = \texttt{insample}'\langle 2\rangle$ hold, as guaranteed by a previous lifting.

Similarly, if $\sigma_i \in \Gamma$ and $\texttt{guard}(r_i) = \texttt{insample} \geq x$, then we can construct the lifting $\sigma_i\langle 1\rangle\{(a,b) : a = \sigma_i \implies b = \sigma_i\}^{\#2d_i}\sigma_i\langle 2\rangle$.

However, if $\sigma_i = \texttt{insample}$ and $\texttt{guard}(r_i) = \texttt{insample} \geq x$, the only lifting we can construct is $\sigma_i\langle 1\rangle\{(a,b) : a = \sigma_i \implies b = \sigma_i\}^{\#\infty}\sigma_i\langle 2\rangle$. If this is the case, we say r_i is *faulty* (in the context of S^L).

Intuitively, with some overhead initial privacy cost, we assign 0 cost to every transition with guard $\texttt{insample} < x$ and twice the standard cost to every transition with guard $\texttt{insample} \geq x$, which is why we call this strategy S^L (for "less than").

Indeed, we have the following transition cost function for $i \in \{1, \ldots, n-1\}$:

$$c^L(t_i) = \begin{cases}
0 & \sigma_i \in \Gamma \land \texttt{guard}(t_i) \neq \texttt{insample} \geq x\\
d_i & \sigma_i = \texttt{insample} \land \texttt{guard}(t_i) \neq \texttt{insample} \geq x\\
d_i' & \sigma_i = \texttt{insample}' \land \texttt{guard}(t_i) \neq \texttt{insample} \geq x\\
2d_i & \sigma_i \in \Gamma \land \texttt{guard}(t_i) = \texttt{insample} \geq x\\
\infty & \sigma_i = \texttt{insample} \land \texttt{guard}(t_i) = \texttt{insample} \geq x\\
2d_i + d_i' & \sigma_i = \texttt{insample}' \land \texttt{guard}(t_i) = \texttt{insample} \geq x
\end{cases}$$

By sequential composition of liftings, we have $\rho_\sigma(X)\{(a,b) : a = \sigma_k \implies b = \sigma_k\}^{\#d_L\varepsilon}\rho_\sigma(X')$, where the total privacy cost d_L^ρ of S^L on path ρ is

$$d_L^\rho = 2d_0 + \sum_{i \in \{1,\ldots,n-1\}} c^L(t_i)$$

Note that for an SSA, $d_L < \infty$ if and only if no transitions in d_L are faulty.

The construction and cost of S^G is exactly symmetric.

Coupling Strategy 3: S^N.

As mentioned, there are occasional restrictions on when S^L and S^G can be used to prove the privacy of a segment. In particular, if $\texttt{trans}(r_0)$ outputs `insample`, or if both a $\texttt{insample} < x$ and a $\texttt{insample} \geq x$ transition output `insample`, then neither S^G nor S^L are valid. For this scenario, we define the coupling strategy S^N as follows.

Fix some $i \in \{0, \ldots, n-1\}$.

For all states r_i, construct the lifting $z_i\langle 1\rangle (=)^{\#(d_i\varepsilon,0)} z_i\langle 2\rangle + \texttt{in}_i\langle 2\rangle - \texttt{in}_i\langle 1\rangle$. This is equivalent to constructing the lifting $\texttt{insample}\langle 1\rangle (=)^{\#d_i\varepsilon} \texttt{insample}\langle 2\rangle$. For $i = 0$, this specifically also is equivalent to constructing the lifting $x\langle 1\rangle (=)^{\#d_0\varepsilon} x\langle 2\rangle$

Again, if $\texttt{trans}(r_i)$ outputs $\texttt{insample}'$, then construct the lifting $\texttt{insample}'\langle 1\rangle (=)^{\#(d_i',0)} \texttt{insample}'\langle 2\rangle$.

As before, we must ensure that the initial transition guard $\texttt{guard}(r_0)$ is satisfied.

We claim that $x_0\langle 1\rangle (=)^{\#0} x_0\langle 2\rangle$, which is sufficient to show that $\texttt{guard}(r_0)$ is satisfied in $A(X) \implies \texttt{guard}(r_0)$ is satisfied in $A(X')$.

We claim that if $\sigma_i \in \Gamma \cup \{\texttt{insample}\}$, we can construct the lifting $\sigma_i\langle 1\rangle \{(a,b) : a = \sigma_i \implies b = \sigma_i\}^{\#d_i\varepsilon} \sigma_i\langle 2\rangle$.

Otherwise, if $\sigma_i = \texttt{insample}'$, then we can construct the lifting $\sigma_i\langle 1\rangle \{(a,b) : a = \sigma_i \implies b = \sigma_i\}^{\#(d_i+d_i')\varepsilon} \sigma_i\langle 2\rangle$

This gives us the following transition cost function for $i \in [n-1]$:

$$c^N(t_i) = \begin{cases} d_i & \sigma_i \neq \texttt{insample}' \\ d_i + d_i' & \sigma_i = \texttt{insample}' \end{cases}$$

Thus by sequential composition of liftings, we have $\rho_\sigma(X)\{(a,b) : a = \sigma_k \implies b = \sigma_k\}^{\#d_N\varepsilon} \rho_\sigma(X')$, where the total privacy cost d_N^ρ of S^N on path ρ is

$$d_N^\rho = \sum_{i \in [n-1]} c^N(t_i).$$

S^N (the "null" coupling strategy) assigns each transition cost proportional to the spread parameter of the Laplace distribution being sampled from at each transition. With S^N, we gain flexibility in outputting `insample` at the cost of every transition being "costly".

4.3.2 Privacy

Definition 4.6. The coupling costs d_N, d_L, d_G for an SSA A are defined as

$$d_N = \sup_\rho d_N^\rho$$
$$d_L = \sup_\rho d_L^\rho$$
$$d_G = \sup_\rho d_G^\rho$$

Proposition 4.7. *If an SSA $A(x)$ has coupling costs d_N, d_L, d_G, $A(x)$ is $\min\{d_N, d_L, d_G\}\varepsilon$-differentially private.*

Proof. For every output σ and associated path ρ, we have shown that the lifting

$$A(x)(X)\{(a,b) : a = \sigma \implies b = \sigma\}^{\# \min\{d_N^\rho, d_L^\rho, d_G^\rho\}\varepsilon} A(x)(X')$$

is valid for adjacent datasets $X \sim X'$. In particular, note that at least d_N^ρ is always finite.

Because there are a finite set of paths through A, d_N, d_L, d_G are always finite if for all ρ, d_N^ρ, d_L^ρ, and d_G^ρ are finite, respectively.

Thus, we can construct the lifting

$$A(x)(X)\{(a,b) : a = \sigma \implies b = \sigma\}^{\# \min\{d_N, d_L, d_G\}\varepsilon} A(x)(X'),$$

which gives us the finite cost $d = \min\{d_N, d_L, d_G\}$. Applying theorem 3.4 then completes the proof. □

It is worth noting that all SSAs are inherently $d\varepsilon$-differentially private for *some* $d > 0$ because there are a finite number of paths through any SSA A.

Thus, we now examine an extension of SSAs that allows for loops in which this is not necessarily the case.

4.4 Loops

Definition 4.8. A looping segment automaton (LSA) $A(x)$ with initial value $x \in \mathbb{R}$ is a tuple $(Q, \Sigma, C, \Gamma, q_{init}, X, P, \delta)$ where $(Q, \Sigma, C, \Gamma, q_{init}, X, P, \delta)$ are defined exactly as in 4.1, **except** that δ has the following, changed set of restrictions:

- **Determinism:** For any state $q \in Q$, if $\delta(q, \texttt{true})$ is defined, then $\delta(q, \texttt{insample} < x)$ and $\delta(q, \texttt{insample} \geq x)$ are not defined.
- **Output Distinction:** For any state $q \in Q$, if $\delta(q, \texttt{insample} \geq x) = (q_1, o_1, b_1)$ and $\delta(q, \texttt{insample} < x) = (q_2, o_2, b_2)$, then $o_1 \neq o_2$ and at least one of $o_1 \in \Gamma$ and $o_2 \in \Gamma$ is true.
- **No Branching:** If $\delta(q, \texttt{insample} < x)$ and $\delta(q, \texttt{insample} \geq x)$ figure out exactly how to define this later

- Also q_{init} can't be in a cycle.

As with SSAs, $A(x)$ is parameterized by some initial real value x, stored in the variable `x` and we will treat x as a random variable drawn from some Laplace distribution.

Definition 4.9. A cycle C in an SSA A is a `G`-cycle if there exists a transition t in C with guard $\texttt{guard}(t) = \texttt{insample} \geq x$. Similarly, C is an `L`-cycle if there exists a transition t in C with guard $\texttt{guard}(t) = \texttt{insample} < x$.

Note that a cycle can be both an `L`-cycle and a `G`-cycle. Also, we will assume that all executions of an LSA A terminate; this means that every cycle in A *must* be either an `L`-cycle or a `G`-cycle.

Paths, path probabilities, and the concept of $d\varepsilon$-differential privacy remain identical for LSAs as for SSAs.

Importantly, the same three coupling strategies for SSAs are still valid proof strategies for LSAs.

Proposition 4.10. *For an LSA A, if at least one of S^N, S^L, or S^G have finite coupling cost on A, then A is $\min\{\varepsilon_N, \varepsilon_L, \varepsilon_G\}$-differentially private.*

The proof of validity for coupling strategies on SSAs holds for LSAs as well.

However, now that LSAs can *fail* to be differentially private, it is worth exploring in exactly what scenarios LSAs will or will not be private, and what couplings have to say about them.

We begin by identifying three graph-theoretic structures that mean that an LSA is not private.

Definition 4.11. A **leaking pair** in an LSA A is a pair of cycles C, C' in A such that there exists a path from C to C', C is an `L`-cycle, and C' is a `G`-cycle(or vice versa).

Definition 4.12. A **disclosing cycle** in an LSA A is a cycle C in A where a transition in C outputs either `insample` or `insample`$'$.

Definition 4.13. A **privacy violating path** in an LSA A is a path $\rho = q_0 \to q_1 \to \ldots \to q_n$ in A such that one of the following conditions holds:

- $q_0 = q_{init}$, $\texttt{trans}(q_0)$ outputs `insample`, and q_n is in a `G`-cycle or an `L`-cycle.
- q_n is in a `G`-cycle(symmetrically, `L`-cycle), $\texttt{guard}(q_0) = \texttt{insample} < x$ (symmetrically, $\texttt{insample} \geq x$), and $\texttt{trans}(q_0)$ outputs `insample`.
- q_0 is in a `G`-cycle(symmetrically, `L`-cycle), $\texttt{guard}(q_n) = \texttt{insample} < x$ (symmetrically, $\texttt{insample} \geq x$), and $\texttt{trans}(q_n)$ outputs `insample`.

Proposition 4.14. *If an LSA A contains a leaking pair, a disclosing cycle, or privacy violating path, then A is not $d\varepsilon$-differentially private for any $d > 0$.*

The proof can be found in [6].

Perhaps surprisingly, we can thus show that if none of the coupling strategies have finite coupling cost for an LSA A, then A is not $d\varepsilon$-differentially private.

Lemma 4.15. *For an LSA A, $d_N < \infty$ if and only if there is no cycle in A. Further, if there exists an* `L`*-cycle in A, then $d_G = \infty$ and if there exists a* `G`*-cycle in A, then $d_L = \infty$.*

Proof. Follows immediately from the coupling strategies cost functions. □

Lemma 4.16. *For an LSA A, there exists a global bound $N_A \in \mathbb{N}$ such that every path ρ in A has at most N_A non-cycle transitions.*

Proof. Follows from the fact that A has a finite number of states. □

Lemma 4.17. *If $d_N = \infty, d_L = \infty$, and $d_G = \infty$, then A must contain a leaking pair, a disclosing cycle, or a privacy violating path.*

Proof. Because $d_N = \infty$, we know there must be some cycle C in A. WLOG suppose that C is an `L`-cycle.

If C is also a `G`-cycle, then C is a leaking pair with itself, so we can assume that C is not also a `G`-cycle. Similarly, if `trans`(q_0) outputs `insample`, then there must be a privacy violating path from q_0 to C, so suppose `trans`(q_0) doesn't output `insample`.

Suppose that there are no disclosing cycles and no leaking pairs in A. Because there are no leaking pairs, a `G`-cycle cannot exist in A. Additionally, because there are no disclosing cycles and no `G`-cycles, every cycle transition t must have $c^L(t) = 0$. Because there are a bounded number of non-cycle transitions in A, d_L can only be ∞ if there exists a faulty transition with respect to S^L in A. Thus, there must be some transition with guard `insample` $\geq x$ that outputs `insample`; the path from or to this transition from C must be a privacy violating path.

Now suppose that there are no privacy violating paths and no leaking pairs in A. If there is no privacy violating path, then there can be no faulty transitions with respect to S^L in A by the reasoning above. Because of the bound on non-cycle transitions in a path in A, $d_L = \infty$ only if there exists some cycle transition t_i in A with non-zero cost. In particular, because there are no leaking pairs, every cycle in A only has transitions with guard `insample` $< x$. In order for such a transition t_i to have cost $c^L(t_i) > 0$, t_i must output either `insample` or `insample`$'$, creating a disclosing cycle.

Finally, suppose that there are no privacy violating paths and no disclosing cycles in A. As before, there can be no faulty transitions with respect to S^L in A and every cycle transition with guard `insample` $< x$ or `true` must have cost 0. Again because of the bound on non-cycle transitions in a path in A, this implies that there must be some cycle transition in A with guard `insample` $\geq x$, creating a `G`-cycle. □

Theorem 4.18. *An LSA A is $d\varepsilon$-differentially private if and only if* $\min\{d_N, d_L, d_G\} < \infty$.

Proof. This follows directly from proposition 4.10, proposition 4.14, and lemma 4.17. □

4.5 Branching

Definition 4.19. A segment automaton (SA) $A(x)$ with initial value $x \in \mathbb{R}$ is a tuple $(Q, \Sigma, C, \Gamma, q_{init}, X, P, \delta)$ where $(Q, \Sigma, C, \Gamma, q_{init}, X, P, \delta)$ are defined exactly as in 4.1 and 4.8, except that δ has the following set of restrictions:

- **Determinism:** For any state $q \in Q$, if $\delta(q, \texttt{true})$ is defined, then $\delta(q, \texttt{insample} < x)$ and $\delta(q, \texttt{insample} \geq x)$ are not defined.
- **Output Distinction:** For any state $q \in Q$, if $\delta(q, \texttt{insample} \geq x) = (q_1, o_1, b_1)$ and $\delta(q, \texttt{insample} < x) = (q_2, o_2, b_2)$, then $o_1 \neq o_2$ and at least one of $o_1 \in \Gamma$ and $o_2 \in \Gamma$ is true.
- q_{init} is not in a cycle.

As before, $A(x)$ is parameterized by some initial real value x, stored in the variable `x` and we will treat x as a random variable drawn from some Laplace distribution.

Once branching gets added into the program model, a single coupling strategy no longer works for the entire automaton; instead, each distinct branch of the automaton will require a possibly different coupling strategy.

Definition 4.20. Consider a path ρ in an SA A. Let $acyclic(\rho)$ be a function that given a path ρ, removes all cycles in ρ; i.e. if $\rho = q_0 \to \ldots \to q_n$, $acyclic(\rho) = q_{x_0} \to \ldots \to q_{x_k}$, where $\{x_i\}_{i=0}^{k}$ is the longest possible subsequence of $[n]$ such that $i \neq j \implies q_{x_i} \neq q_{x_j}$.

Let $\equiv_b$ be an equivalence relation between paths of A where $\rho \equiv_b \rho'$ iff $acyclic(\rho) = acyclic(\rho')$. Then $branches(A)$ is the set of equivalence classes of $\equiv_b$.

Note that there are a finite number of branches for every segment automaton. We aim to assign one coupling strategy to each **branch**. In particular, note that each branch taken in isolation is itself an LSA, so that every segment automaton can be viewed as a combined set of parallel and overlapping looping segment automata.

As before, we can define the cost of each branch under each coupling stategy.

Definition 4.21. Let $A(x)$ be a segment automaton. For a branch $B \in branches(A)$ of A, the coupling costs of S^N, S^L, and S^G are, respectively,

$$d_N^{(B)} = \sup_{\rho \in B} d_N^{\rho}$$
$$d_L^{(B)} = \sup_{\rho \in B} d_L^{\rho}$$
$$d_G^{(B)} = \sup_{\rho \in B} d_G^{\rho}$$

Proposition 4.22. *Consider an SA $A(x)$. If for all branches $B \in branches(A)$ there exists a coupling strategy $S_X^{(B)}$ with finite coupling cost $d_X^{(B)} < \infty$, then $A(x)$ is $d\varepsilon$-differentially private, where*

$$d = \max_{B \in branches(A)} d_X^{(B)}$$

Proof. For every output σ and associated path ρ_σ, we have the lifting

$$A(x)(X)\{(a,b) : a = \sigma \implies b = \sigma\}^{\# \min\{d_N^{(B)}, d_L^{(B)}, d_G^{(B)}\}\varepsilon} A(x)(X'),$$

where $\rho_\sigma \in B \in \mathit{branches}(A)$.

This means that for all outputs σ and adjacent datasets $X \sim X'$,

$$\mathbb{P}[A(X) = \sigma] \le e^{d_X^{(B)}\varepsilon}\mathbb{P}[A(X') = \sigma],$$

where again $\rho_\sigma \in B$ and $d_X^{(B)} = \min\{d_N^{(B)}, d_L^{(B)}, d_G^{(B)}\}$.

This means that for all outputs σ and adjacent datasets $X \sim X'$,

$$\mathbb{P}[A(X) = \sigma] \le e^{\max_{B \in \mathit{branches}(A)}\{d_X^{(B)}\}\varepsilon}\mathbb{P}[A(X') = \sigma],$$

which completes the proof. □

For the exact same reason as LSAs, these coupling strategies indeed still completely characterize the privacy of SAs.

Proposition 4.23. *An SA $A(x)$ is $d\varepsilon$-differentially private if and only if for all branches $B \in \mathit{branches}(A)$ there exists a coupling strategy with finite coupling cost $d_X^{(B)}$.*

Proof. Because each branch of a segment automaton is an LSA, this again follows directly from proposition 4.10, proposition 4.14 (from [6] the same result holds for segment automata), and lemma 4.17. □

5 DiPA

We now introduce the final extension to our program model.

Definition 5.1 ([6])**.** A DiP[9] Automaton (DiPA) A is a 7-tuple $(Q, \Sigma, C, \Gamma, q_{init}, X, P, \delta)$ where

- Q is a finite set of states partitioned into input states Q_{in} and non-input states Q_{non}.
- $\Sigma = \mathbb{R}$ is the input alphabet
- $C = \{\texttt{true}, \texttt{insample} < x, \texttt{insample} \ge x\}$ is a set of guard conditions
- Γ is a finite output alphabet
- $q_{init} \in Q$ is the initial state
- $X = \{\texttt{x}, \texttt{insample}, \texttt{insample}'\}$ is the set of variables
- $P : Q \to \mathbb{Q} \times \mathbb{Q}^{\ge 0} \times \mathbb{Q} \times \mathbb{Q}^{\ge 0}$ is a parameter function that assigns sampling parameters for the Laplace distribution for each state

[9]**Di**fferentially **P**rivate

- $\delta : (Q \times C) \to (Q \times (\Gamma \cup \{\texttt{insample}, \texttt{insample}'\}) \times \{\texttt{true}, \texttt{false}\})$ is a partial transition function.

In addition, δ must satisfy some additional conditions:

- **Determinism:** For any state $q \in Q$, if $\delta(q, \texttt{true})$ is defined, then $\delta(q, \texttt{insample} < x)$ and $\delta(q, \texttt{insample} \geq x)$ are not defined.
- **Output Distinction:** For any state $q \in Q$, if $\delta(q, \texttt{insample} \geq x) = (q_1, o_1, b_1)$ and $\delta(q, \texttt{insample} < x) = (q_2, o_2, b_2)$, then $o_1 \neq o_2$ and at least one of $o_1 \in \Gamma$ and $o_2 \in \Gamma$ is true.
- **Initialization:** The initial state q_0 has only one outgoing transition of the form $\delta(q_0, \texttt{true}) = (q, o, \texttt{true})$.
- **Non-input transition:** From any $q \in Q_{non}$, if $\delta(q, c)$ is defined, then $c = \texttt{true}$.

A DiPA A functions almost identically to the previous automata we have defined, except that instead of only the first transition of A assigning into `x`, now any **assignment transition** can re-assign the value of `x` to be `insample`. In addition, DiPAs are no longer dependent on an initial value for `x`, since the first transition must always be a `true` guard assignment transition.

Note that we can treat a DiPA A as a (possibly infinite, if an assignment transition is in a cycle) set of segment automata by "splitting" at every assignment transition in A.

Definition 5.2. For a path ρ in a DiPA A, let *segments*(ρ) be the set of subpaths of ρ created by splitting $\rho = r_0 \to \ldots \to r_n$ at each assignment transition.

More precisely, let I be the ordered set of indices such that $\texttt{trans}(r_{I_i})$ for $i \in [|I|]$ is an assignment transition. Then *segments*$(\rho) = \{r_0 \to \ldots \to r_{I_0}, \ldots, r_0 \to \ldots \to r_{I_k}\}$.

Definition 5.3. Let A be a DiPA and let P be the set of all paths through A.

Then $\mathit{branches}(A) = \bigcup_{\rho \in P} \bigcup_{\eta \in \mathit{segments}(\rho)} \mathit{acyclic}(\eta)$ be the set of branches in A.

Note that every path in A can be broken up into a sequence of branches, each from its own segment.

5.1 Joining Coupling Strategies

Naively, the fact that we can separate a DiPA A into a set of segment automata would suggest an approach of assigning a coupling strategy to each branch of A as before and simply combining these coupling strategies in sequence. However, it is not always possible to arbitrarily join coupling strategies together.

This is primarily because we can no longer freely decide the initial value for `x` at the beginning of each segment, as we could with segment automata. Recall that for a fixed path $\rho = r_0 \to \ldots \to r_n$, it was important that the initial transition guard $\texttt{guard}(r_0)$ is satisfied. Notably, whether or not the initial guard is satisfied is dependent on the initialized value of `x`; since we had no additional restrictions on `x`, we were able to (for example) arbitrarily couple $x_0\langle 1\rangle + 1 (=)^{\# d_x \varepsilon} x_0\langle 2\rangle$.

However, now that we have joined multiple segment automata together, the initial value x_0 for a segment automata is actually the assigned value `x` from the *previous* segment automata. This means that $x_0\langle 1\rangle$ and $x_0\langle 2\rangle$ will already be coupled together, perhaps in a way such that coupling $\texttt{insample}_0\langle 1\rangle$ and $\texttt{insample}_0\langle 2\rangle$ will be impossible to both ensure the correct shift of `x` for the new coupling strategy and so that the initial guard is satisfied.

For example, consider two paths $\rho = r_0 \to \ldots \to r_n$ and $\rho' = q_1 \to \ldots \to q_m$ with the assignment transition $r_n \to q_1$ connecting them, where $\texttt{guard}(r_n) = \texttt{insample} \geq x$. We are aiming to choose one coupling strategy for each of ρ and ρ'.

If S^L is chosen for ρ, then at the connecting assignment transition, x_0 is coupled such that $x_0\langle 1\rangle + 1 = x_1\langle 1\rangle$. Further, we would like to couple $\texttt{insample}_0$ such that $\texttt{insample}_0\langle 1\rangle = \texttt{insample}_0\langle 2\rangle + 1$. However, trying to create this lifting while also ensuring that the implication $\texttt{insample}_0\langle 1\rangle \geq x_0\langle 1\rangle \implies \texttt{insample}_0\langle 2\rangle \geq x_0\langle 2\rangle$ holds is impossible.

Thus, we cannot choose S^G for a segment that follows a segment with coupling strategy S^L if the connecting assignment transition has guard $\texttt{insample} \geq x$. There are a series of similar restrictions on how coupling strategies can be connected for adjacent segments, which along with previous restrictions on coupling strategies (e.g. we cannot choose S^G or S^L if the first transition of a segment outputs `insample`).

5.2 A Constraint System for Valid Couplings

Based on the restrictions on joining segments together, we can construct a set of constraints on coupling strategies for a DiPA A that, if solved, will give us a valid proof of $d\varepsilon$-differential privacy.

Construction 5.4. Consider a DiPA A.

For each branch $s_i \in \textit{branches}(A)$, we can assign one of three coupling strategies $S_i \in \{S^L, S^G, S^N\}$. We would like to find an assignment of coupling strategies for each segment of each variable, subject to the following constraints:

1. Constraints for valid couplings
   (a) For all s_i, if $\texttt{trans}(s_i)$ outputs `insample`, then $S_i = S^N$.
   (b) For all s_i, s_j such that s_i is directly followed by s_j,
      i. If $\texttt{guard}(s_j) = \texttt{insample} < x$ and $S_i = S^G$, then $S_j = S^G$.
      ii. If $\texttt{guard}(s_j) = \texttt{insample} \geq x$ and $S_i = S^L$, then $S_j = S^L$.
      iii. If $\texttt{guard}(s_j) = \texttt{insample} < x$ and $S_i = S^N$, then $S_j \neq S^L$.
      iv. If $\texttt{guard}(s_j) = \texttt{insample} \geq x$ and $S_i = S^N$, then $S_j \neq S^G$.
2. Constraints for finite cost

(a) For all s_i, no cycle in s_i has a transition that outputs `insample` or `insample`$'$.

(b) For all s_i, if s_i has an `L`-cycle, then $S_i = S^L$.

(c) For all s_i, if s_i has a `G`-cycle, then $S_i = S^G$.

(d) For all segments s_i, there is no transition $\texttt{trans}(a_k)$ in s_i that is *faulty*, i.e.:

i. If s_i contains a `insample` $< x$ transition that outputs `insample`, then $S_i \neq S^G$.

ii. If s_i contains a `insample` $\geq x$ transition that outputs `insample`, then $S_i \neq S^L$.

Theorem 5.5. *Let A be a DiPA. If $|branches(A)| < \infty$ and for each branch $B \in branches(A)$, there exists an assignment of coupling strategies to each segment that satisfies the constraint system defined in 5.4, then A is $d\varepsilon$-differentially private for*

$$d = \sum_{s_i \in branches(A)} d_{s_i},$$

where d_{s_i} is the cost of the satisfying assignment strategy S_i for a branch s_i.

Proof. Let ρ be a path through A that is composed of branches $s_{x_1} \to \ldots \to s_{x_k}$ and let $S_{x_1}, \ldots, S_{x_k}$ be the associated coupling strategies from the satisfying assignment.

Because of constraints (1a-1b), we can actually compose each coupling strategy in sequence to create a valid lifting $A(X)\{(a,b) : a = \sigma \implies b = \sigma\}A(X')$ for all adjacent datasets $X \sim X'$ and possible outputs σ of ρ.

In addition, we claim that the cost of the satisfying assignments d_{s_i} is bounded for all s_i. There are two ways for a branch cost d_{s_i} to be unbounded: either a cycle transition has non-zero cost, or a transition has to be faulty with respect to the chosen coupling strategy. From constraint (2d), we know that no transitions can be faulty, and we know from constraints (2a-2c) that every cycle transition has to have zero cost.

Finally, the fact that $|branches(A)| < \infty$ and every path can go through each branch at most one time gives us the bound on d. □

5.3 Well-formedness

Similar to before, [6] define four structures in the graphs of DiPAs that characterize privacy for DiPAs. Within DiPAs, these graph structures can be defined as follows:

- **Leaking Cycles**: A cycle C is a leaking cycle if one of the transitions in C is an assignment transition.
- **Leaking Pair**: A pair of cycles C, C' are a leaking pair if C is an `L`-cycle, C' is a `G`-cycle, and there exists a path from C to C' such that every assignment transition on the path has guard `insample` $\geq x$; or, symmetrically, C is an `G`-cycle, C' is a `L`-cycle,

and there exists a path from C to C' such that every assignment transition on the path has guard `insample` $< x$.

- **Disclosing Cycle**: A cycle C is a disclosing cycle if a transition in C outputs either `insample` or `insample`$'$.
- **Privacy Violating Path**: A path $\rho = q_0 \to \ldots \to q_n$ is a privacy violating path if any of the following conditions hold:
  - $q_1 \to \ldots q_n$ is a path whose assignment transitions all have guard `insample` $\geq x$ (resp. `insample` $< x$) such that q_n is part of a `G`-cycle (resp. `L`-cycle) and `trans`(q_0) is an assignment transition that outputs `insample`.
  - ρ is a path whose assignment transitions all have guard `insample` $\geq x$ (resp. `insample` $< x$) such that q_n is part of a `G`-cycle (resp. `L`-cycle), `guard`$(q_0) =$ `insample` $< x$ (resp. `insample` $\geq x$), and `trans`(q_0) outputs `insample`.
  - ρ is a path whose assignment transitions all have guard `insample` $\geq x$ (resp. `insample` $< x$) such that q_0 is part of a `L`-cycle (resp. `G`-cycle), `guard`$(q_{n-1}) =$ `insample` $\geq x$ (resp. `insample` $< x$), and `trans`(q_{n-1}) outputs `insample`.

Definition 5.6. A DiPA A is **well-formed** if it does not have a leaking cycle, leaking pair, disclosing cycle, or privacy violating path.

Importantly, [6] have shown that the existence of these graph structures completely decides whether or not a DiPA is $d\varepsilon$-differentially private.

Theorem 5.7 ([6])**.** *A DiPA A is $d\varepsilon$-differentially private for some $d > 0$ if and only if A is well-formed. Further, the well-formedness of an automaton A can be decided in linear time in the size of A.*

We will leverage this result to show that the existence of valid coupling strategies is also a complete characterization of DiPAs.

Lemma 5.8. *For a DiPA A, $|branches(A)| < \infty$ if and only if A has no leaking cycles.*

Lemma 5.9. *Suppose a DiPA A has a finite number of branches. If constraint system 5.4 is not satisfiable, there must exist a leaking cycle, a leaking pair, a disclosing cycle, or a privacy violating path in the graph of A.*

Proof. TBD, want to briefly discuss the best way to do this.

high level overview: very similar to the previous, one-segment, version of this, but we can chain constraints across `AG`$-$ and `AL`$-$paths.

The reasoning gets a bit convoluted though, because there are a bunch of cases / you have to keep a bunch of assumptions hanging around. □

Theorem 5.10. *A DiPA A is $d\varepsilon$-differentially private if and only if A has a finite number of branches and there exists a valid assignment of coupling strategies over all branches of A.*

Proof. Follows from theorem 5.5 and lemmas 5.8 and 5.9. □

6 Bounds on Privacy

So far, we have focused on the binary question of whether a DiPA is private or not for *any* finite $d > 0$. An obvious question remaining thus is how tight of a privacy cost bound can we obtain using couplings?

Recall that S^G, S^L, and S^N are primarily characterized by **shifts**; that is, offseting variables from each other.

Indeed, we can generalize our discrete set of coupling strategies to a continuous family of coupling strategies by treating the offsets of each variable as parameters. By rephrasing in this way, we can derive a linear system:

[insert vishnu's linear program here]

If solvable, this linear system will thus give the best possible privacy cost obtainable using this family of coupling strategies.

Proposition 6.1. *An optimal solution to the linear program provides the minimal cost coupling over all shift coupling strategies. If the linear program is infeasible, then the DiPA is not $d\varepsilon$-differentially private for any ε.*

Theorem 6.2 (optimistically)**.** *If a DiPA A is ε-differentially private, then there exists some valid assignment of coupling strategies over all segments that has a total cost of ε (i.e. coupling cost is tight).*

7 Multivariable DiPA

In this section, we explore a natural extension of the DiPA model to demonstrate the utility of using couplings as proof infrastructure. Specifically, we introduce Generalized DiPAs (GDiPAs), which allow for an arbitrary finite set of variables and an expanded alphabet where multiple variables can be compared to the input simultaneously to determine transitions. This would, for example, allow for SVT-style algorithms that check for membership between or outside of two thresholds or indeed, membership within any finite union of intersections of halves[10] of $\mathbb{R}$.

7.1 GDiPA

A GDiPA A is a generalization of DiPA whose primary characteristics are that:

- At every state, a (real-valued) input is read. Additionally, Laplace noise (with parameters set by the user) is added to the input.

[10]think there's a term for this I'm forgetting

- The automaton has a stored finite set of (real-valued) variables $\mathcal{X}$. At each transition, the value of each variable can be updated with the noisy input value read in.
- A transition can optionally assign the (noisy) input value read at the previous state into at most one program variable.
- The automaton will take transitions based on a boolean combination of comparisons between the noisy input and a subset of the stored variables. Importantly, noise is added independently to the input for each separate variable comparison. Alternatively, there can be exactly one guaranteed (`true`) transition out of a state.
- At every transition, the automaton will output either
  1. A symbol from a pre-defined finite alphabet (Γ)
  2. The noisy input value as compared to one of the input variables (`insample`)
  3. The input value with fresh Laplace noise (`insample`$'$)

  As with DiPAs, the output sequence of any GDiPA must uniquely determine a path through the automaton.

7.2 Combining Separate Variable Couplings

Consider a GDiPA A with program variables $\mathcal{X}$. For each program variable $x \in \mathcal{X}$, we can define coupling strategies similar to with single variables. That is, for two runs of the automaton with adjacent inputs, we must ensure that if the first run takes a certain transition, the other run does as well. In particular, for real outputs, we must ensure that the outputs are equal.

For each variable in A, we can consider a "shadow" automaton in a single variable that only has a single program variable. Such a shadow automaton would have the same underlying graph structure as A, but would only contain the transition guards and assignments that pertained to a single variable. For example, if we were considering such an automaton A_x with respect to the variable x, a transition with guard "`insample` $< x$ and `insample` $\geq y$" would correspond to a transition with guard `insample` $< x$ in A_x; a transition with guard "`insample` $\geq y$" would correspond to a transition with guard `true` in A_x.

Note that these automata are not, strictly speaking, DiPAs, since it is possible for a state to have multiple transitions with guard `true` leaving it.

Thus, each segment can be assigned a coupling strategy S^N, S^L, or S^G as before based on these "shadow" automata, with similar constraints.

Construction 7.1. Consider a GDiPA A with program variables $\mathcal{X}$. Let $\{s_i^{(x)}\}$ be the segments of A for each variable $x \in \mathcal{X}$.

For each $x \in \mathcal{X}$ and segment $s_i^{(x)}$ for x, we can assign one of three coupling strategies $S_i^{(x)} \in \{S^L, S^G, S^N\}$. We would like to find an assignment of coupling strategies for each segment of each variable, subject to the following constraints:

1. Constraints for valid couplings
   (a) For all $s_i^{(x)}$, if $\texttt{trans}(s_i^{(x)})$ outputs $\texttt{insample}$, then $S_i^{(x)} = S^N$.
   (b) For all $s_i^{(x)}, s_j^{(x)}$ such that $s_i^{(x)}$ is immediately followed by $s_j^{(x)}$,
      i. If $\texttt{guard}(s_j^{(x)}) = \texttt{insample} < x$ and $S_i^{(x)} = S^G$, then $S_j^{(x)} = S^G$.
      ii. If $\texttt{guard}(s_j^{(x)}) = \texttt{insample} \geq x$ and $S_i^{(x)} = S^L$, then $S_j^{(x)} = S^L$.
      iii. If $\texttt{guard}(s_j^{(x)}) = \texttt{insample} < x$ and $S_i^{(x)} = S^N$, then $S_j^{(x)} \neq S^L$.
      iv. If $\texttt{guard}(s_j^{(x)}) = \texttt{insample} \geq x$ and $S_i^{(x)} = S^N$, then $S_j^{(x)} \neq S^G$.
   (c) For all segments $s_i^{(x)}$, there is no transition $\texttt{trans}(a_k)$ in $s_i^{(x)}$ that is *faulty*, i.e.:
      i. If $s_i^{(x)}$ contains a $\texttt{insample} < x$ transition that outputs $\texttt{insample}$, then $S_i^{(x)} \neq S^G$.
      ii. If $s_i^{(x)}$ contains a $\texttt{insample} \geq x$ transition that outputs $\texttt{insample}$, then $S_i^{(x)} \neq S^L$.
   (d) If any transition in $s_i^{(x)}$ outputs x, then $S_i^{(x)} = S^N$.
2. Constraints for finite cost
   (a) For all $s_i^{(x)}$, no cycle in $s_i^{(x)}$ has a transition that outputs $\texttt{insample}$, $\texttt{insample}'$.
   (b) For all $s_i^{(x)}$, if $s_i^{(x)}$ has an $\texttt{L}$-cycle with respect to x, then $S_i^{(x)} = S^L$.
   (c) For all $s_i^{(x)}$, if $s_i^{(x)}$ has a $\texttt{G}$-cycle with respect to x, then $S_i^{(x)} = S^G$.

This constraint system is **satisfiable** for a GDiPA A if, for all variables $x \in \mathcal{X}$, there exists an assignment of all $S_i^{(x)}$ such that all constraints are satisfied.

In particular, we can independently resolve constraints for each variable and combine them together.

Theorem 7.2. *For a GDiPA $\mathcal{A}$, for all variables x, if there exist a finite number of segments s_i, and for all segments s_i, the constraint system 5.4 is satisfied by an assignment S_x, then all S_x assignments together induce a valid coupling proof that $\mathcal{A}$ is ε-DP.*

Proof. Fix some variable $x \in \mathcal{X}$. From before, we know that if 7.1 is satisfied with respect to x, then the coupling

$$\mathcal{A}(X)(\mathcal{A}(X) \text{ takes transitions } T \text{ wrt } x \implies \mathcal{A}(X') \text{ takes transitions } T \text{ wrt } x)^{\#(\varepsilon_x, 0)}\mathcal{A}(X')$$

is valid for some finite ε_x.

Because the noises on $\texttt{insample}$ are independent, and $((a \implies c) \wedge (b \implies d)) \implies ((a \wedge b) \implies (c \wedge d))$ and $((a \implies c) \vee (b \implies d)) \implies ((a \vee b) \implies (c \vee d))$,for every combined guard, we can construct the lifting

$$\mathcal{A}(X)(\mathcal{A}(X) \text{ takes transitions } T \implies \mathcal{A}(X') \text{ takes transitions } T)^{\#(\sum_{x \in \mathcal{X}} \varepsilon_x, 0)}\mathcal{A}(X')$$

which gives us the lifting

$$\mathcal{A}(X)(\mathcal{A}(X) = \sigma \implies \mathcal{A}(X') = \sigma)^{\#(\sum_{x \in \mathcal{X}} \varepsilon_x, 0)} \mathcal{A}(X')$$

$\square$

Theorem 7.3 (less optimistically)**.** *If, for each variable x, there exists a valid assignment of coupling strategies over all segments of a DiPA A with respect to x, then A is $d\varepsilon$-differentially private.*

Theorem 7.4 (optimistically)**.** *A GDiPA A is $d\varepsilon$-differentially private if and only if, for each variable x, there exists a valid assignment of coupling strategies over all segments of A with respect to x.*

8 Conclusion

We have shown how to use coupling techniques to prove privacy for a class of SVT-like programs first defined in [6] and discovered that couplings additionally characterize this class. We additionally showed that this can be done tractably, and that couplings can help provide lower bounds on privacy costs of these algorithms.

Future work most naturally would focus on extensions of the program model. For the model, potential areas include removing the requirement for output to be deterministic of a path through the automaton, which would allow for algorithms such as Report Noisy Max to be captured by the model. Similarly, the alphabet of the automaton could be expanded to incorporate more than comparisons between two real numbers. Such extensions would naturally also require extensions of the class of couplings we define here, which are limited to “shifts”.

Additionally, we believe that couplings should completely characterize GDiPAs as well as DiPAs; proving this requires showing that a lack of well-formedness in any single variable generates a counterexample to privacy. In this vein, we would like to explore using couplings to *disprove* privacy; the fact that shift couplings completely characterize DiPAs hints at the possibility of “anti-couplings” to generate counterexamples.

9 Related Work

The DiPA model and counterexamples to privacy are drawn from [6]. Approximate liftings were developed in [5, 4] and applied to algorithms such as SVT in [3]. A full exploration of approximate liftings can be found in [9]. [1] uses couplings; and in particular the “shift” couplings family we use, to create a heuristiccally successful program for proving the correctness of possible differentially private algorithms.

need to reformat some citations at some point

References

[1] Aws Albarghouthi and Justin Hsu. Synthesizing coupling proofs of differential privacy. 58:30, 2018.

[2] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. Deciding differential privacy for programs with finite inputs and outputs. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 141–154, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Proving differential privacy via probabilistic couplings. *Proceedings - Symposium on Logic in Computer Science*, 05-08-July-2016:749–758, 1 2016.

[4] Gilles Barthe and Federico Olmedo. Beyond differential privacy: Composition theorems and relational logic for f-divergences between probabilistic programs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7966 LNCS:49–60, 2013.

[5] Gilles Barthe, Boris K ¨ Opf, Federico Olmedo, and Santiago Zanella-B ´ Eguelin. N probabilistic relational reasoning for differential privacy.

[6] Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. On Linear Time Decidability of Differential Privacy for Programs with Unbounded Inputs, April 2021.

[7] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, aug 2014.

[9] Justin Hsu. Probabilistic couplings for probabilistic reasoning. *CoRR*, abs/1710.09951, 2017.

[10] Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *Proc. VLDB Endow.*, 10(6):637–648, feb 2017.

[11] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and XiaoFeng Wang. Privacy loss in apple's implementation of differential privacy on macos 10.12. *CoRR*, abs/1709.02753, 2017.