Formal Verification of Differential Privacy for Interactive Systems*

Michael Carl Tschantz
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
mtschant@cs.cmu.edu

Dilsun Kaynar
CyLab
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
dilsunk@cmu.edu

Anupam Datta
CyLab
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
danupam@cmu.edu

January 17, 2011

Abstract

Differential privacy is a promising approach to privacy preserving data analysis with a well-developed theory for functions. Despite recent work on implementing systems that aim to provide differential privacy, the problem of formally verifying that these systems have differential privacy has not been adequately addressed. This paper presents the first results towards automated verification of source code for differentially private interactive systems. We develop a formal probabilistic automaton model of differential privacy for systems by adapting prior work on differential privacy for functions. The main technical result of the paper is a sound proof technique based on a form of probabilistic bisimulation relation for proving that a system modeled as a probabilistic automaton satisfies differential privacy. The novelty lies in the way we track quantitative privacy leakage bounds using a relation family instead of a single relation. We illustrate the proof technique on a representative automaton motivated by PINQ, an implemented system that is intended to provide differential privacy. To make our proof technique easier to apply to realistic systems, we prove a form of refinement theorem and apply it to show that a refinement of the abstract PINQ automaton also satisfies our differential privacy definition. Finally, we begin the process of automating our proof technique by providing an algorithm for mechanically checking a restricted class of relations from the proof technique.

*This work was partially supported by the U.S. Army Research Office contract on Perpetually Available and Secure Information Systems (DAAD19-02-1-0389) to Carnegie Mellon CyLab, the NSF Science and Technology Center TRUST, the NSF CyberTrust grant "Privacy, Compliance and Information Risk in Complex Organizational Processes," and the AFOSR MURI "Collaborative Policies and Assured Information Sharing."

1 Introduction

Differential Privacy. Differential privacy is a promising approach to privacy-preserving data analysis (see [Dwo08, Dwo10] for surveys). This work is motivated by statistical data sets that contain personal information about a large number of individuals (e.g., census or health data). In such a scenario, a trusted party collects personal information from a representative sample with the goal of releasing statistics about the underlying population while simultaneously protecting the privacy of the individuals. In an interactive setting, an untrusted data examiner poses queries that the trusted party evaluates over the data set and appropriately modifies to protect privacy before sending the result to the examiner. Differential privacy formalizes this operation in terms of a probabilistic *sanitization function* that takes the data set as input. Differential privacy requires that the probability of producing an output should not change much irrespective of whether information about any particular individual is in the data set or not. The amount of change is measured in terms of a *privacy leakage bound*—a non-negative real number ϵ, where a smaller ϵ indicates a higher level of privacy. The insight here is that since only a limited amount of additional privacy risk is incurred by joining a data set, individuals may decide to join the data set if there are societal benefits from doing so (e.g., aiding cancer research). A consequence and strength of the definition is that the privacy guarantee holds irrespective of the auxiliary information and computational power available to an adversary. Previous work on algorithms for sanitization functions and the analysis of these algorithms in light of the trade-offs between privacy and utility (answering useful queries accurately without compromising privacy) has provided firm foundations for differential privacy (e.g. [DMNS06, Dwo06, MT07, NRS07, BLR08, Dwo08, Dwo09, GRS09, Dwo10, DNPR10]).

In a different direction, these sanitization algorithms are being implemented for inclusion in data management systems. For example, PINQ resembles a SQL database, but instead of providing the actual answer to SQL queries, it provides the output of a differentially private sanitization function operating on the actual answer [McS09]. Another such system, AIRAVAT, manages distributed data and performs MapReduce computations in a cloud computing environment while using differential privacy as a basis for declassifying data in a mandatory access control framework [RRS+10]. Both of these are interactive systems that use sanitization functions as a component: they interact with both the providers of sensitive data and untrusted data examiners, store the data, and perform computations on the data some of which apply sanitization functions. Even if we assume that these systems correctly implement the sanitization functions to give differential privacy, this is not sufficient to conclude that the guarantees of differential privacy apply to the system as a whole. For the differential privacy guarantee of functions to scale to the whole of the implemented system, the system must properly handle the sensitive data and never provide channels through which untrusted examiners can infer information about it without first sanitizing it to the degree dictated by the privacy error bound.

Formal Methods for Differential Privacy. We work toward reconciling formal analysis techniques with the growing body of work on abstract frameworks or implemented systems that use differential privacy as a building block. While prior work in the area has provided a type system for proving that a non-interactive program is a differentially private sanitization function [RP10], we know of no formal methods for proving that an interactive system using such functions has differential privacy. Applying formal methods to interactive systems ensures that these systems properly manage their data bases and interactions with untrusted users.

Formal verification that an interactive system provides privacy requires that the system be modeled in such a way that the correspondence between the system and model is evident and the model includes all relevant behavior of the system. Once formal verification is done on the model, one can assert with the confidence afforded by formal proofs that the system as implemented and modeled preserves privacy in addition to knowing that the algorithms implemented by the system preserve privacy. For formal verification to scale to large programs with complex models, the creation of the model and the verification of its privacy must be mechanized, preferably in a compositional manner.

To this end, we present an automaton model for which the correspondence between the automaton and the implementation of a system is so plainly evident that the automaton could be automatically extracted from source code as is done with model checking [CGP00]. For this model, we introduce a form of compositional reasoning that allows us to separate the proof that a function gives differential privacy from the proof that the system correctly uses that function. Furthermore, we present a proof technique for such models that is amenable to mechanization and an algorithm that can be used to check that the proof technique is correctly applied to a model.

Our effort can be likened to those efforts in the security community that involve the development of formal models for cryptographic protocols and the accompanying verification methods [ST07, BPW07, CCK+08]. These works use stylized proofs with multiple levels of abstraction and compositionality to enable scaling mechanical checking of these proofs to the size of realistic systems. Making these proofs shorter or more readable for humans than their informal counterparts is not a goal.

Contributions. We work with a special class of probabilistic I/O automata that allow us to model interactive systems in terms of states and probabilistic transitions between states. These automata provide us with the needed expressive power for modeling how data is stored in an internal state of an implementation, and how it is updated through computations, some of which apply differentially private sanitization functions on data. In Section 3.1, we present this probabilistic automaton model and our differential privacy definition for probabilistic automata, which we call *differential noninterference* due to the similarities it has with the information flow property *noninterference* [GM82]. Indeed, when applied to interactive systems, both differential privacy and noninterference privacy aim at restricting information leakage about sensitive data by requiring that the system produces similar outputs for inputs that differ only in sensitive data. However, differential privacy allows for the degree of similarity to decrease as the inputs diverge, making it a more flexible requirement.

As formal methods can only scale to large systems with compositional reasoning, in Section 4, we examine the ability to perform compositional reasoning with our formal model. We show that correctness proof of sanitization functions may be separated from the correctness proof of the system that uses them.

Our main technical contribution, presented in Section 5, is a *proof technique* for establishing that a system has differential noninterference. Our technique allows the global property of differential noninterference to be proved from local information about transitions between states. This proof technique was inspired by the *unwinding* proof technique originally developed for proving that a system has noninterference [GM84].

Our unwinding technique is also similar to bisimulation-based proof techniques as both uses a notion of "similarity" of states with respect to their observable behavior. Unlike traditional

bisimulation relations for probabilistic automata, the unwinding relation is defined over the states of a single automaton with the intention of establishing the similarity of two states where one is obtainable from the other by the input of an additional data point. Moreover, the notion of similarity is approximate, which is in keeping with the definition of differential privacy. An unwinding proof involves finding a relation family indexed by the set of possible values of the privacy leakage bound ϵ, rather than a single relation. This departure from traditional probabilistic bisimulations is needed to track the maximum privacy leakage tolerable from a given state in the execution. We prove the soundness of our proof technique in Theorem 2, which roughly states that the existence of appropriate ϵ-unwinding families for an automaton M implies that M has ϵ-differential noninterference.

As in other formal proof techniques of this nature, the real creativity in doing the proofs with our technique goes into defining the unwinding family. Unsurprisingly, the rest consists of repeated, routine applications of basic arguments showing that the defined relation between states is preserved by transitions of the system. In Section 6, this quality enables us to develop an algorithm to check whether a given relation family is an unwinding family, thereby automating proofs for differential noninterference modulo the definition of the relations. We prove that the algorithm soundly runs in polynomial time: it will only return true if the automaton has ϵ-differential noninterference (Theorems 4 and 5).

To motivate our work, we start by presenting a system similar to PINQ. We refer to the example system throughout our paper as we model it in our formalism and use our unwinding technique and algorithm to verify that it has differential noninterference. As PINQ may be configured to use any set of sanitization functions, we present an automaton M_{ex1} that is parametric in the sanitization functions that it uses. We show two methods for proving differential noninterference for any correct instantiation of M_{ex1} with differentially private sanitization functions: by using the composition method presented in Section 4, and by using our unwinding verification algorithm. This second method illustrates the applicability of our algorithm in proving differential noninterference for interesting automata.

Along the way, we find interactions between a bounded memory model and differential privacy of interest beyond formal verification. In particular, we find the inability to store an unbounded number of data points results in doubling the privacy leakage.

We finish with Section 7 covering related work and Section 8 presenting future work and conclusions.

2 Background and Motivation

2.1 Differential Privacy

Differential privacy formalizes the idea that a private process should not reveal too much information about a single person. A *data point* represents all the information collected about an individual (or other entity that must be protected). A multiset (bag) of data points forms a *data set*. A *sanitization function* κ processes the data set and returns a result to the untrusted data examiner that should probabilistically not change whether or not a single data point is in the data set. Dwork [Dwo06] states differential privacy as follows:

Definition 1 (Differential Privacy)**.** *A randomized function κ has ϵ-differential privacy iff for all*

data sets B_1 and B_2 differing on at most one element, and for all $S \subseteq \mathsf{range}(\kappa)$,

$$\Pr[\kappa(B_1) \in S] \leq \exp(\epsilon) * \Pr[\kappa(B_2) \in S]$$

Formally, multisets B_1 and B_2 differ on at most one element iff either $B_1 = B_2$ or there exists d such that $B_1 \cup \{d\} = B_2$ or $B_2 \cup \{d\} = B_1$. Note that the above definition is well-defined only if $\mathsf{range}(\kappa)$ is countable.

Differential privacy has many pleasing properties. For example, if B_1 and B_2 differ by n datapoints instead of just one, then the probabilities of $\kappa(B_1)$ and $\kappa(B_2)$ being in a set S will be within a factor of $\exp(n * \epsilon)$ of one another [MT07, Corollary 4]. Furthermore, a function that sequentially applies n functions each with ϵ-differential privacy and provides all of their outputs is an $(n * \epsilon)$-differentially private function [MT07, Corollary 5].

Privacy Mechanisms. As shown in the original work on differential privacy, given a statistic f that can be computed of the data sets B_i, one can construct a sanitization function κ_f from f by having κ_f add noise to the value of $f(B_i)$ where the noise is drawn from a Laplace distribution [DMNS06]. This is an example of a *privacy mechanism*, a scheme for converting a statistic into a sanitization function with differential privacy.

Systems in practice would implement a sanitization function such as κ_f as a program. As actual computers have only a bounded amount of memory, the program computing κ_f must only use a bounded amount of memory. However, many sanitization functions proposed in the differential privacy literature, including all sanitization functions constructed using the Laplace privacy mechanism, use randomly drawn real numbers, which requires an uncountably infinite number of states. While such functions can be approximated using a finite number of states (e.g., by using floating point numbers), it is unclear whether the proofs that these functions have differential privacy carry over to their approximations.

As we are interested in formally proving that finite systems provide differential privacy, we limit ourselves to privacy mechanisms that operate over only a finite number of values. One such mechanism is the *Truncated Geometric Mechanism* of Ghosh et al. [GRS09], which uses noise drawn from a bounded, discrete version of the Laplace distribution. As we are interested in applying formal methods to systems using such mechanisms, we provide an implementation of this mechanism that runs in expected constant time and proofs about it in Appendix A.

2.2 Motivating Example System

To further motivate and illustrate our work, we provide an example of an interactive system that uses sanitization functions. Throughout the remainder of this paper, we apply the various formal methods we develop to prove that it preserves privacy. The system manages data points entered by data providers and processes requests of data examiners for information by receiving queries and answering them after sanitizing the answer computed over the data set. The system must apply the sanitization functions to the data set and interact with the data examiner in a manner that does not compromise privacy.

Possible source code for one such system is shown in Figure 1. To be concrete, suppose that the data points are integers and the system handles only two queries. The first produces the output of the sanitization function COUNT, which provides the number of data points currently in the data base. The second produces the output of SUM, which provides their sum. In both cases,

```
01 dPts:= emptyArray(t);
02 numPts := emptyArray(t);
03 for(j:=0; j<t; j++)
04   dPts[j]:= emptyArray(maxPts);
05   numPts[j] := 0;
06 curSlot:=0;
07 while(1)
08   y:=input();
09   if(datapoint(y))
10     if(numPts[curSlot]<maxPts)
11       dPts[curSlot][numPts[curSlot]]:=y;
12       numPts[curSlot]++;
13   else
14     k:=get_sanitization_funct(y);
15     res:=k.compute(dPts);
16     print(res);
17     curSlot:=(curSlot + 1) mod t;
18     delete dPts[curSlot];
19     dPts[curSlot] := emptyArray(maxPts);
20     numPts[curSlot] := 0
```

Figure 1: Program that tracks data point usage to ensure differential noninterference

the sanitization functions use the Truncated Geometric Mechanism to preserve privacy [GRS09]. (Appendix A.3 provides source code for COUNT and SUM.)

Intuitively, the program in Figure 1 keeps an array of t arrays of data points and a variable `curSlot`, whose value indicates a (current) slot in the array. If the input is a data point, that data point is added to the array indexed by `curSlot` unless that array is full, in which case the data point is ignored.

If the input is a query, then the query requested by the input is computed on the union of all the data points collected from all the arrays. Line `15` uses either the implementation of COUNT or SUM to compute the system's response to the query `y` where Line `14` selects the correct function. Furthermore, the index `curSlot` to one of these arrays is cyclically shifted and the array to which it now points is replaced with an empty array. Since there are only t slots, this means that each array will only last for t queries before being deleted. (If $t = 0$, we take the program to have an array `dPts` of length 0, in which case it never stores any data points.) Since each query has ϵ-differential privacy, this ensures that each data point will only be involved in $t * \epsilon$ worth of queries.

Verification. The goal of our work is to formally verify that systems like this one preserve the privacy of their users. In addition to showing that the sanitization functions COUNT and SUM have differential privacy (a subject of previous work [GRS09]), we study how the system leaks information about the data points in ways other than through the outputs from these functions. Indeed, one might expect from the sequential result for differential privacy discussed above [MT07, Corollary 5], that the system would provide $(t * \epsilon)$-differential privacy. However, due to how the

system manages data points, it actually only provides $(2t * \epsilon)$-differential privacy as we show later.

Had our goal only been to formally verify the implementations of the sanitization functions COUNT and SUM, it would suffice to use a simple formal model such as that of probabilistic finite-state automata with no interaction and use a suitable algorithmic technique to verify differential privacy, which research on Markov chains provides. (We provide further details in Section 4.1.)

However, to verify differential privacy for interactive systems that use privacy mechanism as a building block as the above system does, we need a more expressive formal model that models the interaction of the data examiner with the system and the addition of data points to the system over time. The next section provides such a model.

3 Modeling Interaction for Formal Verification

In this section, we present the basics of the formal framework we use in modeling interactive systems and show how we can model the example system of Section 2.2 using this formalism. Specifically, in Sections 3.1 and 3.2, we introduce a special class of probabilistic I/O automata and present our definition of differential privacy for this class of probabilistic I/O automata. In Section 3.3 we model the program of Figure 1 as a probabilistic I/O automaton.

3.1 Automata

We use a simplified version of probabilistic I/O automata (cf. [LSV07]). We define an automaton in terms of a probabilistic labeled transition system (PLTS).

Definition 2. *A probabilistic labeled transition system (*PLTS*) is a tuple* $L = \langle S, I, O, \rightarrow \rangle$ *where* S *is a countable set of states;* I *and* O *are countable and pairwise disjoint sets of actions, referred to as* input *and* output *actions respectively; and* $\rightarrow \subseteq S \times (I \cup O) \times \mathsf{Disc}(S)$ *represents the possible transitions where* $\mathsf{Disc}(S)$ *is the set of discrete probability measures over* S*.*

We use A for $I \cup O$. We partition the input set I into D, the set of data points, and Q, the set of queries. We also partition the output set O into R, the set of responses to the data examiner's queries and H, the set of outputs that are hidden from (not observable to) the data examiner. Note that H includes outputs to the data provider. We let E range over all actions to which the examiner has direct access: $E = Q \cup R$. When only one automaton is under consideration, we denote a transition $\langle s, a, \mu \rangle \in \rightarrow$ by $s \xrightarrow{a} \mu$.

Henceforth, we require that PLTSs satisfy the following conditions:

- *Transition determinism:* For every state $s \in S$ and action $a \in A$, there is at most one $\mu \in \mathsf{Disc}(S)$ such that $s \xrightarrow{a} \mu$.
- *Output determinism:* For every state $s \in S$, output $o \in O$, action $a \in A$, and $\mu \in \mathsf{Disc}(S)$, if $s \xrightarrow{o} \mu$ and $s \xrightarrow{a} \mu'$, then $a = o$ and $\mu' = \mu$.
- *Quasi-input enabling:* For every state $s \in S$, inputs i_1 and i_2 in I, and $\mu_1 \in \mathsf{Disc}(S)$, if $s \xrightarrow{i_1} \mu_1$, then there exists μ_2 such that $s \xrightarrow{i_2} \mu_2$.

Output determinism and quasi-input enabling means that the state space may be partitioned into two parts: states that accept all of the inputs and states that produce exactly one output. We

require that each output producing state produces only one output since the choice of output should be made by the PLTS to avoid nondeterminism that might be resolved in a way that leaks information about the data set. Owing to transition determinism, we will often write $s \xrightarrow{a} \mu$ without explicitly quantifying μ.

We define an extended transition relation $\Rightarrow$ that describes how a PLTS may perform a sequence of actions where some of the output actions are hidden from the data examiner. In particular, the hidden outputs in H model unobservable internal actions irrelevant to privacy. To define $\Rightarrow$, let a state that produces an output from H be called *H-enabled* and one that does not be called *H-disabled*. By output determinism, H-enabled states may only transition under an action in H and, thus, cannot have transitions on actions from $R \cup Q \cup D$. To skip over such states and focus on H-disabled states, which are more interesting from a verification point of view, we define $\Rightarrow$ to show to which H-disabled states the system may transition while performing any finite number of hidden actions. We define $s \stackrel{a}{\Rightarrow} \nu$ so that $\nu(s')$ is the probability of reaching the H-disabled state s' from the state s where a is the action performed from state s. Note that ν is not a distribution over the set S of states since the automaton might execute an infinite sequence of H-enabled states never reaching an H-disabled state. We let ν be a distribution over $S_\perp = S \cup \{\perp\}$ where $\perp \notin S$ represents nontermination and $\nu(\perp) = 1 - \sum_{s \in S} \nu(s)$. Note that for no a, μ, or ν does $\perp \xrightarrow{a} \mu$ or $\perp \stackrel{a}{\Rightarrow} \nu$.

A PLTS L combined with a state s defines a probabilistic I/O automaton $\langle L, s \rangle$. This state is thought of as the initial state of the automaton or the current state of the PLTS. We define a *trace* to be a sequence of actions from $A^* \cup A^\omega$. Given such an automaton M, we define $[\![M]\!]$ to be a function from input sequences to the random variable over traces that describes how the automaton M behaves under the inputs $\vec{i}$. We let $\lfloor [\![M]\!](\vec{i}) \rfloor_E$ denote the random variable over sequences of actions observable to the data examiner obtained by projecting only the actions in E from the trace returned by random variable $[\![M]\!](\vec{i})$.

To deal with nontermination, we note that the examiner can only observe finite prefixes of any nonterminating trace. When the examiner sees the finite prefixes of a trace, he must consider all traces of the system with the observed prefix as possible. (The set of these traces has been called a *cone* — see e.g. [LSV07].) Since the examiner may only see actions in E, these sets are in one-to-one correspondence with E^*. Thus, the examiner observing some event is not modeled as the probability of the system producing a trace in some set, but rather with the probability of a system producing a prefix of trace in some set. That is, rather than using $\Pr[\lfloor [\![M]\!](\vec{i}) \rfloor_E \in S]$ for $S \subseteq E^* \cup E^\omega$, we need $\Pr[\lfloor [\![M]\!](\vec{i}) \rfloor_E \sqsupseteq S]$ for $S \subseteq E^*$ where $\sqsupseteq$ is the super-sequence-equal operator raised to work over sets in the following manner: $\vec{e} \sqsupseteq S$ iff there exists $\vec{e}' \in S$ such that $\vec{e} \sqsupseteq \vec{e}'$ where $\vec{e} \in E^* \cup E^\omega$ and $S \subseteq E^*$.

In Appendix B, we formalize these concepts and show how to calculate these probabilities from the transitions of the automaton.

3.2 Differential Noninterference

Often the data set of a differentially private system is loaded over time and may change between queries. Such changes in the data set are not explicitly modeled by the definition of differential privacy, but one could conceive of modeling such changes by having data points be time-indexed sequences of data. Nevertheless, for formal verification, we require an explicit model of data set mutation. Thus, we present a version of differential privacy defined in terms of the behavior of an automaton that accepts both queries and data points over time.

Definition 3 (Differential Noninterference)**.** *An automaton M has ϵ-differential noninterference if for all input sequences $\vec{i}_1$ and $\vec{i}_2$ in I^* differing on at most one data point, and for all $S \subseteq E^*$,*

$$\Pr[\lfloor [\![M]\!](\vec{i}_1)\rfloor_E \sqsupseteq S] \leq \exp(\epsilon) * \Pr[\lfloor [\![M]\!](\vec{i}_2)\rfloor_E \sqsupseteq S]$$

where we say two input sequences differ by one data point if one of the sequences may be constructed from the other by inserting a single data point anywhere in it.

By restricting the traces of M to only those elements of $E = Q \cup R$, we limit traces to only those actions accessible to the untrusted data examiner. The definition requires that any subset of such traces be almost equally probable under the input sequences $\vec{i}_1$ and $\vec{i}_2$, which differ by at most one data point. Note that like the original form of differential privacy, we do not model the adversary explicitly but rather consider the behavior of the automaton over all possible input sequences the adversary could supply.

In Appendix C, we give full definitions for sequence differencing and prove results showing that our adaptation of differential privacy preserves pleasing properties of the original. One such property is a composition result (Proposition 13): the privacy leakage bound for a system whose inputs differ on at most n data points is $n * \epsilon$ where ϵ is the leakage bound for the system if its inputs differ on one data point.

3.3 Example: Automaton Model for Program of Figure 1

To eventually prove that the program of Figure 1 has $(2t * \epsilon)$-differential noninterference, we first give an automaton model of the program, called $M_{\mathsf{ex1}}(K)$. Note that the model we give here is parametric in the set of sanitization functions; it applies not only to the program of Figure 1, which assumes $K = \{\textsc{count}, \textsc{sum}\}$ but to any other instance of the same program that uses a possibly different set of sanitization functions (modeled by the parameter K). We define below the state space S and transition relation $\rightarrow$, which determine $L_{\mathsf{ex1}}(K) = \langle S, I, O, \rightarrow\rangle$ for every set K of sanitization functions. Using an initial state s_0, we get the automaton $M_{\mathsf{ex1}}(K) = \langle L_{\mathsf{ex1}}(K), s_0\rangle$.

States. Each state of the automaton can be viewed as a particular valuation of the variables in the program allowed by its type. We model the array `dPts` as a t-tuple of multisets of data points. We model `numPts` as a t-tuple of integers ranging from 0 to v where v is the value held by the constant `maxPts`. We model the index `curSlot` as an integer c ranging from 0 to $t-1$, which selects one of the multisets of the t-tuple. The variable `y` stores the most recent input. The variable `res` keeps track of which output from O is about to be produced and the sanitization function is stored in `k`. The state must also keep track of a program counter pc, which ranges over the program line numbers from 01 to 20. Thus, the set of states S is $\{01, \ldots, 20\} \times (\mathsf{bag}(D))^t \times \{0, \ldots, v\}^t \times \{0, \ldots, t-1\} \times I \times O \times K$ where $\mathsf{bag}(D)$ is the set of all multisets with elements from D and K is the set of sanitization functions.

Actions. We model the `input` command in the source code with the input action set I of our automaton: for each possible value that `input` can return there is an input action in I corresponding to that value. Inputs in the code can be either queries or data points, which is modeled by the partition of the set I into the sets Q for queries and D for data points. We model the `print` command in the source code with the observable outputs R (responses) of our automaton. For each possible value that can be printed we have an output action in R. We model all other commands by internal (hidden) actions.

Transitions. We list below only those transitions that are interesting for our purposes. That is, transitions on actions from the sets I and R, and transitions on hidden actions that represent internal computation such as choosing of an appropriate sanitization function for a given query and computation of the result using that function. We use the symbol τ for hidden actions. We also use *Dirac* distributions: let $\mathsf{Dirac}(s)$ be the distribution such that $\Pr[\mathsf{Dirac}(s){=}s] = 1$ and $\Pr[\mathsf{Dirac}(s){=}s'] = 0$ for all $s' \neq s$. Given a query q in Q, we let κ_q be the sanitization function that answers that query. Some key transitions are:

Input $\langle 08, \vec{B}, \vec{n}, c, y, r, k\rangle \xrightarrow{i} \mathsf{Dirac}(\langle 09, \vec{B}, \vec{n}, c, i, r, k\rangle)$

Choose Function $\langle 14, \vec{B}, \vec{n}, c, y, r, k\rangle \xrightarrow{\tau} \mathsf{Dirac}(\langle 15, \vec{B}, \vec{n}, c, y, r, \kappa_y\rangle)$

Compute Function $\langle 15, \langle B_0, \ldots, B_{t-1}\rangle, \vec{n}\rangle, c, y, r, k\rangle \xrightarrow{\tau} \mu$ where

$$\mu(\langle 16, \langle B_0, \ldots, B_{t-1}\rangle, \vec{n}, c, y, r', k\rangle) = \Pr[k(\biguplus_{\ell=0}^{t-1} B_\ell) = r']$$

using $\uplus$ for multiset union and $\mu(s') = 0$ for states not of that form, and

Output Result $\langle 16, \vec{B}, \vec{n}, c, y, r, k\rangle \xrightarrow{r} \mathsf{Dirac}(\langle 17, \vec{B}, \vec{n}, c, y, r, k\rangle)$

The third transition above is a probabilistic transition that represents the internal computation of a sanitization function k on the union of the multisets $B_0, \ldots, B_{t-1}$. The effect of the transition is to update the value of the *pc* from 15 to 16 and to update the result to be output from r to a new value r' such that the probability of ending up in state $\langle 16, \langle B_1, \ldots, B_t\rangle, c, n, y, r', k\rangle$ as a result of the transition is $\Pr[k(\biguplus_{\ell=1}^{t} B_\ell) = r']$.

From these transitions, we can calculate the extended transitions for each of the three types of H-disabled states:

Drop $\langle 08, \vec{B}, \vec{n}, c, y, r, k\rangle \stackrel{d}{\Rightarrow} \mathsf{Dirac}(\langle 08, \vec{B}, \vec{n}, c, d, r, k\rangle)$ when n_c of $\vec{n}$ is v;

Add $\langle 08, \vec{B}, \vec{n}, c, y, r, k\rangle \stackrel{d}{\Rightarrow} \mathsf{Dirac}(\langle 08, \vec{B}', \vec{n}', c, d, r, k\rangle)$ when n_c of $\vec{n}$ is less than v and $\vec{B}'$ and $\vec{n}'$ are such that $B'_c = B_c \uplus \{d\}$, $n'_c = n_c + 1$, and for all $c' \neq c$, $B'_{c'} = B_{c'}$ and $n'_{c'} = n_{c'}$;

Answer Query $\langle 08, \langle B_0, \ldots, B_{t-1}\rangle, \vec{n}, c, y, r, k\rangle \stackrel{q}{\Rightarrow} \nu$ where

$$\nu(\langle 16, \langle B_0, \ldots, B_{t-1}\rangle, \vec{n}, c, q, r', \kappa_q\rangle) = \Pr[k(\biguplus_{\ell=0}^{t-1} B_\ell) = r']$$

and $\nu(s') = 0$ for states not of that form; and

Delete Old Data $\langle 16, \vec{B}, \vec{n}, c, y, r, k\rangle \stackrel{r}{\Rightarrow} \mathsf{Dirac}(\langle 08, \vec{B}', \vec{n}, c, y, r, k\rangle)$
where we have $B'_{c+1 \mod t} = \{\!\{\}\!\}$, $n'_{c+1 \mod t} = 0$, and for all $c'' \neq c+1 \mod t$, $B'_{c''} = B_{c''}$ and $n'_{c''} = n_{c''}$ using $\{\!\{\}\!\}$ for the empty multiset.

The third extended transition above represents a sequence of transitions that starts with the input of a query q. The input of the query is followed by transitions on hidden actions that model the computation of the answer to the query where some of these hidden steps are probabilistic. The resulting state has the property that κ_q has been chosen as the sanitization function and that

$pc = 16$, which implies that the resulting state is H-disabled and the automaton is ready to perform an observable output by outputing the answer to the query.

The state space S and transition relation $\rightarrow$ determines the PLTS $L_{\mathsf{ex1}}(K) = \langle S, I, O, \rightarrow \rangle$ for every set K of differentially private functions. Using the initial state $s_0 = \langle 1, \{\!\{\}\!\}^t, 0^t, 1, y_0, r_0, k_0 \rangle$, we get the automaton $M_{\mathsf{ex1}}(K) = \langle L_{\mathsf{ex1}}(K), s_0 \rangle$. (The initial values y_0, r_0, k_0 do not matter since they will be replaced before being used.)

Verification of Differential Privacy and Bounded Memory. The remainder of this paper develops the proof techniques needed to formally verify that models such as the one shown above has differential noninterference. In particular, in the next section, we describe a composition result that allows to separately consider whether the sanitization functions in K have differential privacy and whether M_{ex1} properly uses them. In Section 5, we present a proof technique using *unwinding families* for showing that for all sets K of sanitization functions with ϵ-differential privacy, the automaton $M_{\mathsf{ex1}}(K)$ has $(2t * \epsilon)$-differential noninterference. Lastly, Section 6, provides a proof-checking algorithm that ensures our unwinding technique is properly used. These methods together allow for a compositional and mechanically checked formal proof of differential noninterference.

Given that the system modeled above uses ϵ-differentially private functions t times, one might be surprised that we prove that it has $(2t * \epsilon)$-differential noninterference rather than $(t * \epsilon)$-differential noninterference. This extra leakage comes from dealing with the bounded memory of actual computers. In particular, each array in `dPts` is limited to a length of `maxPts`. The program keeps track of the current number of data points stored in each slot with the array `numPts`. If the current slot has reached `maxPts` data points, the program drops any incoming data points until `curSlot` advances.

This dropping of data points introduces extra privacy leakage. A single data point can have two effects: it is both included in calculations and can cause the system to drop future data points and exclude from calculations. Thus, the system has only $(2t * \epsilon)$-differential noninterference. In many scenarios, the possibility of running out of memory for storing data points is unrealistic. If the number of data points can never reach the memory bound, then under this assumption, one can show that system has $(t * \epsilon)$-differential noninterference.

It may be tempting to use a linked list for each slot and keep track of how many total data points are stored in all the slots combined. Then, the program could drop data points only when all the memory is exhausted instead of just the current slot's allocation. However, this change would allow a single data point stored in one slot to affect which data points are dropped from other slots in the future. Thus, a single data point may have an unbounded effect on future computation preventing such a program from satisfying differential noninterference for any privacy bound.

4 Decomposing Verification

Recall the example system presented in Section 2.2. The source code in Figure 1 is written parametrically in the set of sanitization functions (Lines `14` and `15`). The model $M_{\mathsf{ex1}}(K)$ of the system given in Section 3.3 is parametrized over the set of sanitization functions K where the computation of a sanitization function from K is idealized as a single transition in the transition system of $M_{\mathsf{ex1}}(K)$. We will call such models in which computation of functions are abstracted as a single step *idealized models*. In reality, any function in the set K would be implemented by a subroutine that can be modeled by an automaton and an *implementation model* could be obtained from an ideal-

ized model by replacing each idealized transition for a sanitization function with its corresponding subroutine automaton.

In this section, we first provide an algorithm for checking that such subroutine automata modeling sanitization functions have differential privacy. Second, we show how to use the proof that a subroutine has differential privacy to simplify the task of proving that an interactive system using that function has differential noninterference. That is, we show how we support compositional reasoning by separating the verification of a sanitization function from the verification of a system that uses the function.

4.1 Mechanized Verification of Differential Privacy

Previous work has provided a method of formally verifying that a sanitization function has differential privacy [RP10]. Their method operates over a special language to enable type-checking. Below we provide an alternative using automata to model the function.

In particular, we model a subroutine implementing a sanitization function k operating on the database B using an I/O automaton $M_{k,B}$. As k performs no I/O, the model $M_{k,B}$ has an empty set of inputs and only one output h, a hidden action. The initial state of $M_{k,B}$ represents the start of the computation k operating on the argument B. For each output r in the range of k, $M_{k,B}$ has a terminal state $\xi(r)$ with no outgoing transitions corresponding to returning the value r. Since k is a function, $s_0 \stackrel{h}{\Rightarrow} \nu$ must be a distribution over these terminal states with $\nu(\bot) = 0$ and $\nu(s) = 0$ for all states not corresponding to an output.

A function k has ϵ-differential privacy only if M_{k,B_1} and M_{k,B_2} induces sufficiently close distributions over related terminal states for all data bases B_1 and B_2 differing by at most one data point. In particular, for all r in the range of k, $\nu_1(\xi_1(r)) \leq \exp(\epsilon) * \nu_2(\xi_2(r))$ where ν_i is the distribution over terminal states induced by the automaton M_{k,B_i} and ξ_i is the mapping from the range of k to terminal states for M_{k,B_i}.

Thus, mechanically checking if a function k has differential privacy reduces to constructing the appropriate models M_{k,B_i}, computing the distributions ν_i for each of them, and comparing them as needed. As we are only concerned with systems that can actually be implemented, only a finite number of models and comparisons are needed. The construction of the models may be done using known techniques from model checking (see, e.g., [CGP00]). The most complex step is computing the distributions ν_i.

Fortunately, each of these automaton models M_{k,B_i} may be converted to an *absorbing Markov chain*, a model of random behavior leading to one of a fixed set of *absorbing states* each representing a different outcome. Under this conversion, the probability of the Markov chain leading to a particular absorbing state corresponds to the distribution ν_i over terminal states of M_{k,B_i}. This conversion starts with finding the set S' of all H-disabled states reachable from s_0 by using hidden actions. For this task, we may view the transition system as a directed graph G where the nodes are states. If $s \stackrel{h}{\rightarrow} \mu$ and $\mu(s') > 0$ for some hidden action h, then we add an edge from s to s' labeled with $\mu(s')$ to G. (Recall that s will never transition under more than one such hidden action due to the transition-determinism axiom.) A depth-first search may then find those states reachable from s in G. Second, we remove all states from G that are not reachable from s. Third, we convert all states in G that are reachable from s that do not reach any H-disabled states to a single state $s_\bot$, which we treat as an H-disabled state. We can do this with a reachability analysis for each state to every H-disabled state. Forth, we add a self-loop labeled with probability 1 from

every H-disabled state (including $s_\perp$) to itself. The resulting graph corresponds to an absorbing Markov chain where the H-disabled states (including $s_\perp$) are the absorbing states.

To compute the absorbing probabilities of the H-disabled states, we use the standard method as presented in [GS97]. First, we represent the chain using a transition matrix $\mathbf{P}$ in *canonical form*. That is, we renumber the states so that the non-absorbing, or *transient*, states come first in $\mathbf{P}$. In our case, these are the H-enabled states. Let t be the number of transient states and r be the number of absorbing states. We may view $\mathbf{P}$ as having the following form:

$$\mathbf{P} = \left[\begin{array}{c|c} \mathbf{Q} & \mathbf{R} \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right]$$

where $\mathbf{Q}$ is a t-by-t matrix, $\mathbf{R}$ is a non-zero t-by-r matrix, $\mathbf{I}$ is a r-by-r identity matrix, and $\mathbf{0}$ is a r-by-t zero matrix. Here, $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{I}$ capture, the probabilities for, respectively, moving from a transient state to a transient state, moving from a transient state to an absorbing state, and moving from an absorbing state to an absorbing state. Second, from $\mathbf{P}$, we compute *fundamental matrix* $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$. Third, we compute $\mathbf{A} = \mathbf{NR}$. The entry a_{ij} of $\mathbf{A}$ is the probability of the chain ending in (being absorbed by) the state numbered j when started in the state i. Thus, we may set $\nu(s') = a_{ij}$ where i is the number of the initial state and j is the number of the state s'. We refer the reader to [GS97] for the correctness of this algorithm for computing the absorbing probabilities.

Algorithm `closure`(M, s, a)**.** The above algorithm may be generalized to compute ν for a state s and an action a where $s \stackrel{a}{\Rightarrow} \nu$. The generalization replaces initial state with s and constructs the terminal absorbing states from the H-disabled states reachable from s. Let `closure`(M, s, a) denote the generalized algorithm used this way to compute ν such that $s \stackrel{a}{\Rightarrow} \nu$.

As for the runtime of `closure`, note that the first step of constructing of the graph G runs in $O(|S|)$ where S is the state space of M_{k,B_i}. Converting G to use $s_\perp$ takes $O(|S|^2)$. Every other step of the conversion process runs in $O(|S|)$. The matrix operations used to compute the matrix $\mathbf{A}$ can all be done in $O(|S|^3)$ as $t \leq |S|$ and $r \leq |S|$. Thus, it runs in $O(|S|^3)$ time. Using `closure` for computing each ν_i, we may check if k has differential privacy in $O(m + |\mathcal{B}| * |D| * |S|^3)$ where m is the time required to compute all the models and $\mathcal{B}$ is the set of all databases B_i.

4.2 Implementation and Composition

The ability to verify that a subroutine provides differential privacy aids the verification that a system using that subroutine has differential noninterference. In particular, this section shows that the verification of differential noninterference may assume that the subroutine provides a differentially private distribution over return values in a single idealized transition, without modeling the internal transitions of the subroutine. Doing the verification based on such an idealized model is more manageable than doing it based on a model that includes the details about the implementation of the subroutine.

Implementing a Transition with an Automaton. We now define what it means in our model for a single step transition on a hidden action to be implemented by an automaton with a series of hidden transitions. We base our notion of implementation on hidden transitions since it is sufficiently general for our purposes — we do not concern ourselves with the general question of

preserving all kinds of observable behavior through implementation but rather the more restricted question of preserving the resulting distribution over computed values.

A single internal transition of an automaton M_1 may result in a distribution over next states that corresponds to the distribution over terminal states induced by many internal transitions in another automaton M_2. To formalize this, let $s^\dagger$ be a state of the automaton M_1 such that $s^\dagger \xrightarrow{h^\dagger}_1 \mu^\dagger$ for some hidden action $h^\dagger$ of M_1. Let ι be an injection from $\mathsf{Supp}(\mu^\dagger)$ to the state space of some other automaton M_2 such that every state in the image of ι is disabled for every action (i.e., they are terminal states). We say that the automaton M_2 *implements the transition of* $s^\dagger$ *under* ι if for all $s \in \mathsf{Supp}(\mu^\dagger)$, $\mu^\dagger(s) = \sum_{\vec{h} \in H_2^+} M_2([\,])(\vec{h}, \iota(s))$ where H_2 is the hidden action set of M_2 and H_2^+ is the set of non-empty finite sequences using elements from H_2. That is, M_2 implements the transition of $s^\dagger$ under ι if the distribution over the terminal states that M_2 reaches is isomorphic to $\mu^\dagger$ under ι.

Subroutine Composition. Subroutine composition can be viewed as replacing a single step transition in an idealized model with its automaton implementation where such repeated replacements can be used to derive an implementation model from the idealized model.

Let $M_1[s^\dagger, M_2, \iota]$ denote the automaton that results from replacing an internal transition from the state $s^\dagger$ of M_1 with the subroutine M_2 with the injection ι providing how to return from the subroutine. Formally, given $M_1 = \langle\langle S_1, Q_1 \uplus D_1, R_1 \uplus H_1, \rightarrow_1\rangle, s_1^0\rangle$, $M_2 = \langle\langle S_2, \emptyset, H_2, \rightarrow_2\rangle, s_2^0\rangle$, $s^\dagger \in S_1$ such that $s^\dagger \xrightarrow{h^\dagger}_1 \mu^\dagger$ for some hidden action $h^\dagger \in H_1$ and $\mu^\dagger$ where $s^\dagger$ is the unique state that enables $h^\dagger$, $s^\dagger \notin \mathsf{Supp}(\mu^\dagger)$, and $\iota : \mathsf{Supp}(\mu^\dagger) \to S_2$ such that every state in its image is disabled for all actions, let $M_1[s^\dagger, M_2, \iota]$ denote the automaton $M_3 = \langle\langle S_1 \uplus S_2, I_1, R_1 \uplus H_1 \uplus H_2 \uplus \{h^\ddagger\}, \rightarrow_3\rangle, s_1^0\rangle$ where $\uplus$ is disjoint union and $\rightarrow_3$ is defined as follows:

- $s_1 \xrightarrow{a}_3 \mu$ if $s_1 \in S_1$, $s_1 \neq s^\dagger$, and $s_1 \xrightarrow{a}_1 \mu$;
- $s_2 \xrightarrow{a}_3 \mu$ if $s_2 \in S_2$ and $s_2 \xrightarrow{a}_2 \mu$;
- $s^\dagger \xrightarrow{h^\ddagger}_3 \mathsf{Dirac}(s_2^0)$; and
- $\iota(s_1) \xrightarrow{h^\ddagger}_3 \mathsf{Dirac}(s_1)$ for all $s_1 \in \mathsf{Supp}(\mu^\dagger)$.

The special hidden action $h^\ddagger$ in the definition of M_3 above is used to mark the entry and exits points of the subroutine represented by M_2. This extra action is used to correctly "hook up" M_2 with M_1 to obtain M_3.

The lemma below states that if some internal transition of an automaton M_1 (for example, a step corresponding to calling a sanitization function in a differentially noninterference system) is replaced by an automaton M_2 (for example, multiple steps corresponding to a subroutine implementing the sanitization function), then the observable behavior of the resulting automaton is identical to that of M_1.

Theorem 1 (Subroutine Composition)**.** *For all automata M_1 and M_2, states $s^\dagger$, and injections ι such that M_2 implements the transition of $s^\dagger$ under ι, for all $\vec{i}$ in I^*, and $\vec{e}$ in E^*,*

$$\Pr[\,\lfloor M_1(\vec{i})\rfloor_E \sqsupseteq \vec{e}] = \Pr[\,\lfloor M_1[s^\dagger, M_2, \iota](\vec{i})\rfloor_E \sqsupseteq \vec{e}]$$

In Appendix D, we prove this by way of two lemmas.

A corollary is that if an idealized model has differential noninterference then a implementation model formed by replacing its internal transitions with subroutine automata also has differential noninterference.

4.3 Example: Decomposing Verification

Suppose that M_{ex2} is the automaton obtained from $M_{\mathsf{ex1}}(\{\text{COUNT}, \text{SUM}\})$ by replacing the transitions that represent the computations of the functions COUNT and SUM with subroutine automata M_{COUNT,B_i} and M_{SUM,B_i}. That is, M_{ex2} is the code shown in Figure 1 with the implementations of COUNT and SUM in-lined. We may apply the composition theorem repeatedly for each replacement of a single transition in $M_{\mathsf{ex1}}(\{\text{COUNT}, \text{SUM}\})$ with a subroutine automaton in M_{ex2}. Such repeated compositions reduces the problem of verifying differential noninterference for M_{ex2} to two smaller problems: First, we must show that the automata M_{COUNT,B_i} and M_{SUM,B_i} implement with a series of internal transitions the transitions corresponding to the functions COUNT and SUM found in $M_{\mathsf{ex1}}(\{\text{COUNT}, \text{SUM}\})$ as described in our formal definition of *implementation*. Second, we must show that the idealized model $M_{\mathsf{ex1}}(\{\text{COUNT}, \text{SUM}\})$ has the differential noninterference.

The first problem can be solved using `closure`, which establishes that the automaton correctly implement COUNT and SUM. As COUNT and SUM has differential privacy (proofs provided in Appendix A), we may conclude that these subroutine automata have differentially private distributions over their terminal states.[1] The next two sections deal with solving the second problem.

While COUNT and SUM are simple sanitization functions, the above approach generalizes to more complex sanitization functions: As long as the function can be modeled as a series of internal transitions that ends in states corresponding to its return values, our approach will apply. While most of the algorithms previously published use unbounded state spaces, we believe our approach can handle bounded versions of them.

5 Unwinding Proof Technique

We desire a technique for drawing conclusions about the global behavior (executions) of the system from local aspects (states, actions, and transitions) of the model. Faced with a similar situation, Goguen and Meseguer introduced unwinding relations to simplify proving that a system has noninterference [GM84]. We present a similar technique for proving that a system has differential noninterference. In particular we state what it means for a relation family to be an unwinding family and prove Theorem 2, which roughly states that the existence of an unwinding family for a given automaton implies that it satisfies differential noninterference. Our unwinding notion is probabilistic and approximate, which is in keeping with the notion of differential privacy. The novelty lies in the way we keep track of the privacy leakage bound, which evolves as the system evolves where the evolution is constrained by the differential privacy definition.

5.1 Definition and Soundness

Formulating a notion of unwinding relation that is sound for showing differential noninterference is more complicated than existing notions for showing noninterference because we must deal with

[1] We may also mechanically prove that these subroutine automata have differential privacy using other formal methods such a type system [RP10].

probabilities and we must keep track of the privacy leakage bound ϵ. To deal with probabilities and approximation, we adapt the notion of *approximate lifting* from previous work on approximate probabilistic simulation relations in the context of cryptographic protocols [ST07]. However, such work does not deal with tracking a leakage bound (see Section 7 for additional details). Thus, we introduce a *family* of unwinding relations indexed by various amounts of privacy leakage. Each unwinding relation in the family is a relation on the state space of the automaton. The unwinding relation indexed by the leakage amount ϵ relates states that exhibit approximately the same trace distributions in the sense of ϵ-differential noninterference.

To deal with probabilities in a concise and modular way, we first define an approximate lifting operation that takes a relation over sets and produces a relation over distributions on those sets. The degree of approximation is governed by a parameter δ.

Definition 4 (δ-Approximate Lifting)**.** *Let* R *be a relation between a set* X *and a set* Y*. The* δ-approximate lifting *of* R *denoted by* $\mathcal{L}(\mathsf{R}, \delta)$ *is the relation between* $\mathsf{Disc}(X)$ *and* $\mathsf{Disc}(Y)$ *such that for all* ν_1 *in* $\mathsf{Disc}(X)$ *and* ν_2 *in* $\mathsf{Disc}(Y)$*,* ν_1 $\mathcal{L}(\mathsf{R}, \delta)$ ν_2 *if and only if there exists a bijection* $\beta : \mathsf{Supp}(\nu_1) \to \mathsf{Supp}(\nu_2)$ *such that for all* x *in* $\mathsf{Supp}(\nu_1)$*,* x R $\beta(x)$ *and* $|\ln \nu_1(x) - \ln \nu_2(\beta(x))| \leq \delta$*.*

The requirement for β to be from the support set of ν_1 to the support set of ν_2 ensures that if a state is assigned a non-zero probability in ν_1 then it is not possible for a related state to be assigned a zero probability in ν_2 and vice versa—there is one to one correspondence between the states with non-zero and identical probabilities in the two distributions. The form of δ involves natural logarithms because the privacy leakage bound in the differential privacy definition appears in the exponent.

Next we define our unwinding technique, which is illustrated in Figure 2. Intuitively, since we want the behavior of the automaton to change only by a factor of ϵ on receiving a single data point, we want the transitions under a data point from a state s to lead to states s' that are only a factor of ϵ different from s. *Covering* (Definition 6) formalizes this by requiring that state s is related to each such state s' by a relation $\mathcal{R}^\epsilon$ that is part of an ϵ-*unwinding family* (Definition 5).

In more detail, an ϵ-unwinding family starts with a privacy leakage budget of ϵ, which decreases over time to a current balance of ϵ'. Related states s_1 and s_2 are required to only make transitions under the same actions. The distributions ν_1 and ν_2 that result from these transitions followed by any number of transitions under hidden outputs may differ only by a factor of δ. This difference is subtracted from the current balance ϵ' to get a new current balance. Once the balance reaches zero, the resulting distributions must be equivalent. As the balance started at ϵ, only a total of ϵ privacy can be leaked, a point proved in Lemma 1.

Definition 5 (ϵ-Unwinding Family)**.** *For a non-negative real number* ϵ*, a family indexed by the set* $[0, \epsilon]$ *of relations* $\mathcal{R}^{\cdot}$ *over the* H*-disabled states of a* PLTS L *is an* ϵ-unwinding family *for* L *if for all* ϵ' *in* $[0, \epsilon]$*, for all* x_1 *and* x_2 *in* $S_\perp$ *such that* x_1 $\mathcal{R}^{\epsilon'}$ x_2*, for all* a *in* $I \cup R$*, there exists* ν_1 *such that* $x_1 \stackrel{a}{\Rightarrow} \nu_1$ *iff there exists* ν_2 *such that* $x_2 \stackrel{a}{\Rightarrow} \nu_2$*, and when they do exist, there exists a real number* δ *in* $[0, \epsilon']$ *such that* ν_1 $\mathcal{L}(\mathcal{R}^{\epsilon'-\delta}, \delta)$ ν_2*.*

Lemma 1. *For all* ϵ*-unwinding families* $\mathcal{R}^{\cdot}$*, all* ϵ' *in* $[0, \epsilon]$*, all* x_1 *and* x_2 *in* $S_\perp$ *such that* x_1 $\mathcal{R}^{\epsilon'}$ x_2*, all* $\vec{i}$ *in* I^**, and all* $\vec{e}$ *in* E^**, both*
$\Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon') \Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}]$ *and*
$\Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon') \Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}]$*.*

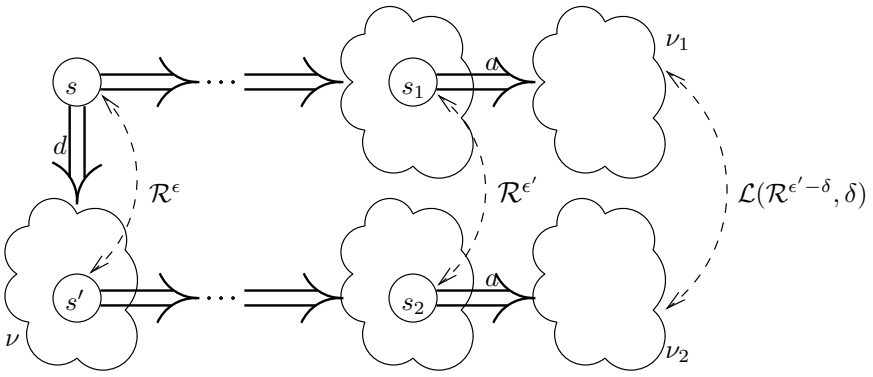

Figure 2: Unwinding Family and Covering: The left side shows the requirements for a covering. The right side shows the requirements placed on an unwinding family. The solid arrows denote the extended transition relation $\Rightarrow$ and clouds depict probability distributions such as ν where $s' \in \mathsf{Supp}(\nu)$.

The above lemma shows that two states related by an ϵ-unwinding family, given the same input sequence, produce distributions that only deviate by a factor ϵ. Thus, to maintain ϵ-differential noninterference, we desire that a state s should upon receiving a single data point d transition to a state s' that can be put into an ϵ-unwinding family with s. We formalize this intuition with the next definition and confirm it with the following theorem.

Definition 6 (Covers)**.** *We say that an ϵ-unwinding family $\mathcal{R}^{\cdot}$ for a* PLTS L covers *a state s and data point d of L if $s \stackrel{d}{\Rightarrow} \nu$ implies that $\nu(\bot) = 0$ and for all $s' \in \mathsf{Supp}(\nu)$, $s\ \mathcal{R}^\epsilon\ s'$.*

Theorem 2. *For an automaton $M = \langle L, s_0 \rangle$, if for all H-disabled states s reachable from s_0 and all data points d, there exists a ϵ-unwinding family that covers s and d, then $[\![M]\!]$ has ϵ-differential noninterference.*

Appendix E holds the proofs of Lemma 1 and Theorem 2. We prove Lemma 1 by induction over the structure of $\vec{a}$. The interesting cases arise when $\vec{a}$ is of the form $i{:}\vec{a}'$ for $i \in I$ or $o{:}\vec{a}'$ for $o \in O$, which require similar reasoning. Suppose that $\vec{a} = i{:}\vec{a}'$ and $s_1 \stackrel{i}{\Rightarrow} \nu_1$ for some $i \in I$. By the unwinding relation, we know that there exists a transition $s_2 \stackrel{i}{\Rightarrow} \nu_2$ such that ν_1 and ν_2 are in keeping with the privacy leakage bound imposed by the unwinding relation. Then for states $s_1' \in \mathsf{Supp}(\nu_1)$, and $s_2' \in \mathsf{Supp}(\nu_2)$, we apply the inductive hypothesis for $\vec{a}'$ to obtain the result.

To prove Theorem 2, we use Proposition 12 and show for all $\vec{i}_1$, $\vec{i}_2$, and $\vec{e}$ where $\Delta(\vec{i}_1, \vec{i}_2) = 1$ that $\Pr[\lfloor [\![\langle L, s\rangle]\!](\vec{i}_1)\rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon) \Pr[\lfloor [\![\langle L, s\rangle]\!](\vec{i}_2)\rfloor_E \sqsupseteq \vec{e}]$. We use proof by induction over $\vec{i}_1$, $\vec{i}_2$, and $\vec{e}$. When we reach the point where $\vec{i}_1$ and $\vec{i}_2$ differ by a data point d, we apply Lemma 1 knowing that an ϵ-unwinding family exists for the current state s and d.

5.2 Example: Applying the Proof Technique

We now return to the parametric automaton model $M_{\mathsf{ex1}}(K)$ of Section 3.3. We show that for any K, every state s and data point d of $L_{\mathsf{ex1}}(K)$ can be *covered* by a $(2t * \epsilon)$-unwinding family $\mathcal{R}^{\cdot}_{s,d}$ in the sense of Definition 6. Differential noninterference will follow from Theorem 2.

For the $(2t * \epsilon)$-unwinding family $\mathcal{R}^{\cdot}_{s,d}$, we construct for each j in $[0, t]$ the unwinding relation $\mathcal{R}^{2j*\epsilon}_{s,d}$. To construct these unwinding relations, we first introduce some notation.

For a state $s = \langle pc, \vec{B}, \vec{n}, c, y, r, k\rangle$ and $d \in D$, $\mathsf{add}(s, c', d)$ adds d to the slot c' of the state s. Formally,

$$\mathsf{add}(s, c', d) = \langle pc, \vec{B}', \vec{n}, c, y, r, k\rangle$$

where $\vec{B}' = \vec{B}$ and $\vec{n}' = \vec{n}$ when $n_c = v$ and, otherwise, $B'_c = B_c \uplus \{d\}$, $n'_c = n_c + 1$, and for all $c' \neq c$, $B'_{c'} = B_{c'}$ and $n'_{c'} = n_{c'}$.

The function swap replaces one data point with another. Formally,

$$\mathsf{swap}(s, c', d, d') = \langle pc, \vec{B}', \vec{n}, c, y, r, k\rangle$$

where $B'_{c'} = B_{c'} - \{d'\} \uplus \{d\}$ and $B'_{c''} = B_{c''}$ for all $c'' \neq c'$.

For j such that $0 \leq j \leq t$, let S^j_1 to be the set of all states s_1 such that s_1 is reachable from s using $t - j$ queries and any number of data points. Intuitively, this means that from s_1 one can pose j more queries until the privacy budget runs out on the data point that is input into the system in state s. We define the relations as follows:

- For $j > 0$, let $\mathcal{R}^{2j*\epsilon}_{s,d}$ to be such that for all $s_1 \in S^j_1$, $s_1\ \mathcal{R}^{2j*\epsilon}_{s,d}\ \mathsf{add}(s_1, c, d)$ and for all d', $s_1\ \mathcal{R}^{2j*\epsilon}_{s,d}\ \mathsf{swap}(s_1, c, d, d')$ where $s = \langle pc, \vec{B}, \vec{n}, c, y, r, k\rangle$. That is, $\mathcal{R}^{2j*\epsilon}_{s,d}$ relates a state to the states it could have become had it received d as input when the `curSlot` was c, the value `curSlot` had in state s.
- For $j = 0$, $\mathcal{R}^{2j*\epsilon}_{s,d}$ is as above for states with a PC of 16 and is equality for those with a PC of 08.

Lemma 2. *For all sets K of functions such that each function in K has ϵ-differential privacy, for all states s and for all data points d, $\mathcal{R}^{\cdot}_{s,d}$ is a $(2t*\epsilon)$-unwinding family for the automaton $M_{\mathsf{ex1}}(K)$.*

Appendix F holds the proof. The proof uses a case analysis over the different types of actions a that might be received by two related states. The most interesting case is when a is a query and $j = 1$. In this case, $s_1\ \mathcal{R}^{\epsilon'}_{s,d}\ s_2$ implies that s_1 is in S^{t-1}_1 with s_1 and s_2 reached in $t-1$ queries. For a $2t*\epsilon$ privacy leakage bound, this corresponds to the last time d may be used in answering a query. This requirement is met since for s_1 and s_2 to be reached with $t-1$ queries, by the construction of $M_{\mathsf{ex1}}(K)$, `curSlot` in both states must be $t-1$ slots away from the slot that holds d. Thus, after answering the next query the slot `curSlot`, whose value is always mod t, will point to the slot that holds d and that slot will be rewritten removing d.

Since $\mathcal{R}^{2j*\epsilon}_{s,d}$ covers s and d for all states s and data points d of the automaton $M_{\mathsf{ex1}}(K)$, Lemma 2 and Theorem 2 implies that the automaton has $(2t * \epsilon)$-differential noninterference.

Theorem 3. *For all set of functions K such that each function in K has ϵ-differential privacy, $M_{\mathsf{ex1}}(K)$ has $(2t * \epsilon)$-differential noninterference.*

As COUNT and SUM are ϵ-differentially private functions, this implies that $M_{\mathsf{ex1}}(\{\text{COUNT}, \text{SUM}\})$ has $(2t * \epsilon)$-differential noninterference. Furthermore, as explained in Section 4.3, subroutine composition shows that M_{ex2}, a system with COUNT and SUM implemented as subroutines instead of atomic transitions, has $(2t * \epsilon)$-differential noninterference. Thus, we have proved that our example has $(2t * \epsilon)$-differential noninterference. In the next section we turn to mechanically verifying differential noninterference.

```
isUnwindFam(⟨S, I, O, T⟩, rel, δ, t)
  convert all hidden actions of ⟨⟨S, D, Q, R, T⟩, s0⟩ to be the same one
  if(|rel| ≠ t + 1),
    return false
  for all i in [0, t],
    for all ⟨x1, x2⟩ ∈ rel[i],
      for all a ∈ I ∪ O,
        if (T[x1][a] = nil xor T[x2][a] = nil),
          then return false
        if (T[x1][a] ≠ nil and T[x2][a] ≠ nil),
          ν1 = closure(⟨S, I, O, T⟩, x1, a)
          ν2 = closure(⟨S, I, O, T⟩, x2, a)
          if(not isInLiftedRelation(S⊥, rel[i], 0, ν1, ν2))
            if(i = 0),
              return false
            if(not isInLiftedRelation(S⊥, rel[i − 1], δ, ν1, ν2))
              return false
  return true
```

Figure 3: Algorithm for checking relation families.

6 Mechanizing Verification of Unwinding

We provide an algorithm that soundly checks if a given family of relations is an unwinding family for a given automaton. While our algorithm does not generate the unwinding family, it automates the process of showing that a candidate family satisfies all the conditions for being an unwinding family (Definition 5). By repeatedly applying our algorithm to a collection of relation families, we can algorithmically check that the covering condition of Theorem 2 holds and that automaton has differential noninterference. The process of verifying an unwinding relation family manually is typically tedious and sometimes error-prone. The existence of a mechanized verifier hence adds practical value to the proof technique presented in the previous section and justifies its use in favor of ad hoc proof methods.

6.1 Algorithm

Our algorithm `isUnwindFam` takes as input a labeled transition system of finite size, an array of relations over the system's states, a value δ, and a natural number t. The array `rel` may only represent relation families $\mathcal{R}^{\cdot}$ over the interval $[0, t * \delta]$ of a restricted form. $\mathcal{R}^{\cdot}$ must be such that $\mathcal{R}^{j\delta} = \mathcal{R}^{k\delta}$ for all j and k such that $\lfloor j \rfloor = \lfloor k \rfloor$. That is, it must be possible to break the index set of $\mathcal{R}^{\cdot}$ into t intervals of size δ such that the relations in that interval are the same and one point corresponding to $\mathcal{R}^{t*\delta}$.

The algorithm is shown in Figure 3. It represents the transition relation $\rightarrow$ as an array T with $|S_\perp|$ rows and $|A|$ columns where $T[\perp][a] =$ `nil` for all a. The array either stores a distribution over next states or `nil` to indicate that the state cannot transition under that action.

The first step of the algorithm converts all the hidden actions to be the same one since `closure` presumes just one hidden action. The function `closure`, defined in Section 4.1, computes the

```
isAllCovered(⟨⟨S, I, O, T⟩, s_0⟩, Rels, δ, t)
   reachableStates := computeReachableStates(⟨S, D, Q, R, T⟩,s_0)
   for all s in reachableStates,
      for all d ∈ D,
         if(T[s][d] ≠ nil),
            ν = closure(⟨S, I, O, T⟩, s, d)
            if(ν(⊥) ≠ 0 or |Rels[s][d]| ≠ t + 1),
               return false
            for all s' ∈ S,
               if(ν(s') > 0 and ⟨s, s'⟩ ∉ Rels[s][d]),
                  return false
            if(not isUnwindFam(⟨S, I, O, T⟩, Rels[s][d], δ)
               return false
   return true
```

Figure 4: Algorithm for checking for differential privacy.

distribution over states that results from the system exhibiting the observable behavior a from a state x_i and computing until reaching an H-disabled state.

The distributions resulting from `closure` are compared with the provided family `rel` using the function `isInLiftedRelation` to determine whether they obey the requirements of a unwinding family. `isInLiftedRelation`$(\mathsf{R}, \delta, \nu_1, \nu_2)$ checks if the two distributions ν_1 and ν_2 are related by the δ-approximate lifting of R. This function operates in $O(|S|^{2.5})$ time by reducing the problem to the decision problem of if a perfect matching exists for a bipartite graph, which can be solved in $O(|S|^{2.5})$ using the Hopcroft-Karp algorithm [HK73]. The reduction constructs a bipartite graph such that each vertex in the left part of the graph corresponds to a state in the support of ν_1, and each in the right part to a state in the support of ν_2. Edges connect those states x_1 in the left part to those x_2 in the right part such that $x_1 \ \mathsf{R} \ x_2$ and $|\ln \nu_1(x_1) - \ln \nu_2(x_2)| \leq \delta$. A matching of graph that includes every vertex (i.e., a *perfect* matching) exists iff there is a bijection showing that $\nu_1 \ \mathcal{L}(\mathsf{R}, \delta) \ \nu_2$. Appendix G formally presents the algorithm and proves this result.

The following lemmas state, respectively, the soundness and the runtime complexity of the algorithm. Appendix H contains the proofs for this section.

Lemma 3 (Soundness)**.** *If the algorithm* `isUnwindFam`$(L, \texttt{rel}, \delta, t)$ *returns true, then* `rel` *corresponds to relation family that is* $(t * \delta)$*-unwinding family for* L*.*

Lemma 4 (Runtime Complexity)**.** *The algorithm* `isUnwindFam` *runs in* $O(t * |A| * |S|^4)$ *time.*

We use `isUnwindFam` to construct an algorithm `isAllCovered` that checks a collection of relation families to conclude if they prove that an automaton has differential privacy (using Theorem 2). In particular, the algorithm takes as input an automaton, an array `Rels` of relation families, δ, and the natural number t. For all states s that are reachable from the start state of the automaton and data points d, the algorithm uses `isUnwindFam` to check whether `Rels`$[s][d]$ corresponds to a $(t * \delta)$-unwinding family that covers s and d. The algorithm is shown in Figure 4.

The following theorems state the soundness and the runtime complexity of the procedure for checking whether all reachable states are covered by a given collection of relation families.

Theorem 4 (Soundness)**.** *If* `isAllCovered`$(M, \texttt{Rels}, \delta, t)$ *returns true, then* M *has* $(t{*}\delta)$*-differential noninterference.*

Theorem 5 (Runtime Complexity)**.** *The algorithm* `isAllCovered` *runs in* $O(t * |D| * |A| * |S|^5)$ *time.*

While sound, the algorithm is not complete even for this restricted class of unwinding relations it accepts as input. The algorithm (soundly) rejects any family if it has a relation that relates two states that transition to distributions over next states that differ by more than δ. That is, it requires that the automaton never leaks more than a δ worth of private information in a single step. Furthermore, it pessimistically presumes that every leakage of private information is a whole δs worth.

Nevertheless, we believe the algorithm is still of interest. In the next section, we show that it is powerful enough to prove that our example system, which is similar to PINQ, has differential noninterference. While this system only has two very simple sanitization functions, COUNT and SUM, our algorithm will work for more complex sanitization functions provided they can be computed with a finite number of states.

6.2 Example: Using the Algorithm

To use our algorithm, we must first model the above program as an automaton M_{ex2} with the subroutines COUNT and SUM in-lined as explained in Section 4.3. Then, we must construct `Rels`, which stores all the needed $(2t * \epsilon)$-unwinding families in the correct format. Such families exist since whenever M_{ex2} leaks privacy, it leaks no more than $2 * \epsilon$ in a single step, and, thus, we can use $2 * \epsilon$ for δ. These families are instances of the parametric families shown in Section 5.2. The reader can confirm that these families may be expressed in the needed format for `Rels`.

Indeed, as the body of the sanitization functions consists entirely of H-enabled states, only the distributions over return values matter to our algorithm in that they influence the computation of `closure` and nothing more. Thus, the general families further shows that our algorithm can verify any modification of M_{ex2} that substitutes a different set of ϵ-differentially private functions for $\{\text{COUNT}, \text{SUM}\}$ provided that those functions can be implemented using a bounded number of states as we would expect from the discussion of composition in Section 4.3.

7 Related Work

Formal Verification of Differential Privacy. The most closely related work to ours is a programming language with a linear type system for proving that well-typed programs in the language have differential privacy [RP10]. Later work applies their type system to detecting network attacks in a private manner [RAW+10]. The usual trade-offs between a program analysis technique designed to work over standard programming languages and a custom type system for a specialized language apply: the type system makes explicit in the source code why the program has differential privacy and type checking scales well, but the programmer must use a special-purpose programming language and annotate the code as the type system requires. Additionally, their programming language lacks I/O commands for creating interactive systems whereas our proof technique is for automata modeling interactive systems.

Other Differential Privacy Definitions. The definition of differential privacy may be seen as largely a simplification of the previously defined notion of ϵ-*indistinguishability* [DMNS06], which explicitly models interaction between a private system and the data examiner as in our definition of differential noninterference. Our definition, however, is cast in the framework of probabilistic automata rather than Turing machines. This supports having structured models that are capable of highlighting issues arising from the bounded memory of actual computers. Furthermore, we deal with non-termination using prefixes allowing us to leverage previous work on formal methods for automata (e.g., [LSV07]).

Differential privacy is a very active research field giving rise to new definitions and techniques at a fast pace [Dwo10, DNPR10]. For example, *pan-privacy* is a notion of differential privacy that gives differential privacy against adversaries that can observe the internal state of a system, in addition to outputs [MPRV09]. *Computational differential privacy* gives certain differential privacy guarantees against computationally bounded adversaries. Our definition of differential noninterference and the formal proof technique was developed from the definition of Dwork [Dwo06]. We think that our choice of probabilistic automata as a model would prove useful in extending the work of this paper to these new definitions as well. For example, algorithms such as stream-processing algorithms that have been subject to research from pan-privacy point of view can be naturally modeled using probabilistic automata. Similarly, probabilistic automata-based models have successfully been used in the formal analysis of cryptographic protocols against computationally bounded adversaries [ST07, BPW07, CCK$^+$08].

Information-Flow Properties. Differential noninterference has some similarities with information flow properties such as noninterference [GM82]. The literature contains several works on the use of transition systems, observational equivalences, and various notions of bisimulation relations to define information flow properties. To name a few, Focardi and Gorrieri have developed a classification of noninterference-like properties in the unifying framework of a process algebra in a non-probabilistic setting [FG01]. Sabelfeld and Sands [SS00], and Smith [Smi03] have used probabilistic bisimulation in defining probabilistic noninterference for multi-threaded programs, which they enforce using type systems. Probabilistic noninterference is regarded by many to be too strong in practice since it requires the probabilities of traces of the system observable by low-level users to be identical for any pair of high-level inputs (data points in our setting) [Gra91, Gra92]. As noninterference is often too strong of a requirement, weaker probabilistic versions have been proposed that allow for some information leakage [PHW04, BP02]. Di Pierro, Hankin, and Wiklicky introduced *approximate noninterference* [PHW04], and Backes and Pfitzmann introduced *computational probabilistic noninterference* [BP02], both of which allow for some information leakage. However, unlike differential noninterference, they do not allow the system behavior to diverge as the difference between the high-level inputs (data points) increases. This divergence, which is allowed by our differential noninterference definition (Proposition 13 in Appendix C), is needed to release meaningful statistics and gain utility from the data set as discussed in detail in Section 1.

Quantitative information flow analysis attempts to determine how much information a program provides an adversary about a sensitive input or class of inputs. Clark, Hunt, and Malacaria present a formal model of programs for quantifying information flows and a static analysis that provides lower and upper bounds on the amount of information that flows [CHM07]. They measure information flow as the mutual information between the high-level inputs and low-level outputs given that the adversary has control over the low-level inputs. Malacaria extends this work to

handle loops [Mal07], and Chen and Malacaria to multi-threaded programs [CM07]. McCamant and Ernst [ME07], and Newsome and Song [NS08] provide dynamic analyses for quantitative information flow using the mutual information formalization. There is also recent work on efficient computation of information leakage in the information theoretic-sense using a probabilistic automaton model [APvRS10]. All of the above approaches assume that the adversary's beliefs are aligned with the actual distribution producing the sensitive input(s) and that adversary has no additional background knowledge. Clarkson, Myers, and Schneider instead propose a formulation using the beliefs of the adversary [CMS05]. However, such a formulation may be difficult to apply in practice because the surveyor may not know the adversary's beliefs. An advantage of differential privacy is that no assumptions are needed about the adversary's auxiliary information, computational power, or beliefs.

Proof Techniques for Transition Systems. Simulation and bisimulation provide a systematic proof technique for showing implementation and equivalence relationships between two automata [Mil89, LV95, SL95] and are related to unwinding (see e.g., [BFPR03]). Most similar to our unwinding technique, Segala and Turrini have studied approximate simulation relations in the context of cryptographic protocols [ST07]. Their work differs from ours by using asymptotic approximations and only executions of polynomial length in terms of a security parameter. Their work allows certain transitions of the protocol to not have a matching transition in the specification. This models the capability of the adversary to compromise correctness. A protocol is deemed correct if the leakage accumulated at the end of a polynomial length execution is exponentially small in some security parameter. Our unwinding technique, on the other hand, requires that there always be an approximately matching transition, uses an exact error bound, and considers executions of any length. However, the probabilities of those transitions are only within some exponential multiplicative factor of one another. Thus, neither approach subsumes the other. Furthermore, our relations are over states whereas theirs is over prefixes of executions.

Much work has been done on decision algorithms for probabilistic simulation and bisimulation [BHK04, BEMC00, PLS00, CS02]. Particularly relevant are the works of Baier and Hermans [BHK04], and Cattani and Segala [CS02] on decision algorithms for weak bisimulations. Since our unwinding relations keep track of an error bound in the form of indices in a relation family, the methods of these papers to generate relations do not readily apply to our setting. We limit ourselves to checking if a given relation family is an unwinding family rather than generating one. Extending these prior works to our setting remains as future work.

Finding refinement methods that preserve information flow properties has been investigated by several authors [Man01, JÖ1, HPS01, AvZ06]. In most of those works refinement is used in the sense of reducing various flavors of nondeterminism in an abstract system. For example, Mantel focuses on a range of information flow properties and unwinding conditions as local conditions that imply these properties [Man01]. He then presents some operators that refine a given transition system such that these conditions are preserved in the system refined by the given operators. We have a more restricted goal in this paper, namely, to pin down the conditions under which an abstract internal transition can be replaced by a sequence of internal transitions in a way that will preserve differential noninterference. This is sufficient for our purposes because such transition replacements are the sources of different abstraction levels that typically arise in the analysis of systems we consider in this paper.

8 Future Work

The results of this paper represent progress towards developing a basis for the formal verification of differential privacy for systems, but leave open several interesting directions that we plan to explore in future work. We hope to create a decision procedure for our proof technique by extending prior work on decision procedures for probabilistic bisimulations [BHK04, BEMC00, PLS00, CS02] to make them produce a family of relations rather than a single one. We also plan to extend the theory to model and reason about higher level systems, such as computer systems of hospitals and other distributed systems [RRS+10] that allow interactions of the system with data providers and with data analysts, while protecting the privacy of the data stored and manipulated by the system. For example, AIRAVAT allows computations over data distributed in a cloud, and combines mandatory access control with differential privacy where differential privacy is used to facilitate declassification governed by the privacy error bound set by a data provider. Our techniques can currently apply to the verification of differential privacy property of the AIRAVAT system using a whole-system model. We are interested in exploring the computational model of AIRAVAT further to understand the interplay between the fine-grained access control mechanisms and the differential privacy mechanisms in stating the end-to-end information-flow guarantee of AIRAVAT. Moreover, we wish to extend compositionality aspects of our framework so that we can decompose the reasoning about such properties, and exploit our proof technique for differential noninterference for parts of the proof. Finally, while the current paper uses manually constructed automata models of systems, we plan to develop techniques to extract such models from source code of software systems such as PINQ [McS09] and AIRAVAT [RRS+10].

Acknowledgments. We thank Jeremiah Blocki and Michael Dinitz for helping us understand infinity.

References

[APvRS10] Miguel E. Andres, Catuscia Palamidessi, Peter van Rossum, and Geoffrey Smith. Computing the leakage of information-hiding systems. In *Proceedings of Sixteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *LNCS*, pages 373–389. Springer, 2010.

[AvZ06] Rajeev Alur, Pavol Černý, and Steve Zdancewic. Preserving secrecy under refinement. In *Proceedings of 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 107–118, 2006.

[BEMC00] C. Baier, B. Engelen, and M. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60:187–231, 2000.

[BFPR03] Annalisa Bossi, Riccardo Focardi, Carla Piazza, and Sabina Rossi. Bisimulation and unwinding for verifying possibilistic security properties. In *VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 223–237, London, UK, 2003. Springer-Verlag.

[BHK04] C. Baier, H. Hermanns, and J.-P. Katoen. Probabilistic weak simulation is decidable in polynomial time. *Information Processing Letters*, 89(3):123–152, 2004.

[BLR08] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to non-interactive database privacy. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 609–618, New York, NY, USA, 2008. ACM.

[BP02] Michael Backes and Birgit Pfitzmann. Computational probabilistic non-interference. In *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security*, pages 1–23, London, UK, 2002. Springer-Verlag.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability framework for asynchronous systems. *Information and Computation*, 2007. Preprint on IACR ePrint 2004/082.

[CCK+08] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Time-bounded task-pioas: A framework for analyzing security protocols. *Journal of Discrete Event Dynamic Systems*, 18(1):111–159, 2008. Short version appeared In 20th Symposium on Distributed Computing (DISC), 2006.

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.

[CHM07] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15:321–371, 2007.

[CM07] Han Chen and Pasquale Malacaria. Quantitative analysis of leakage for multi-threaded programs. In *PLAS '07: Proceedings of the 2007 workshop on Programming languages and analysis for security*, pages 31–40, New York, NY, USA, 2007. ACM.

[CMS05] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*, pages 31–45, Washington, DC, USA, 2005. IEEE Computer Society.

[CS02] Stefano Cattani and Roberto Segala. Decision algorithms for probabilistic bisimulation. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *CONCUR '02: Proceedings of the 13th International Conference on Concurrency Theory*, volume 2421 of *LNCS*, pages 371–385. Springer, 2002.

[DMNS06] Cynthia Dwork, Frank Mcsherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[DNPR10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *In Proceedings of the 42nd ACM Syposium on the Theory of Computing (STOC)*, 2010.

[Dwo06] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, volume 2, pages 1–12, 2006.

[Dwo08] Cynthia Dwork. *Theory and Applications of Models of Computation*, volume 4978, chapter Differential Privacy: A Survey of Results, pages 1–19. Springer, 2008.

[Dwo09] Cynthia Dwork. The differential privacy frontier (extended abstract). In *6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 496–502. Springer, 2009.

[Dwo10] C. Dwork. Differential privacy in new settings. In *Proceedings of Symposium on Discrete Algorithms (SODA)*. SIAM, 2010.

[FG01] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (Part I: Information flow), 2001.

[GM82] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, page 11. IEEE, 1982.

[GM84] Joseph A. Goguen and Jose Meseguer. Unwinding and inference control. In *Proc. of IEEE Symp. on Security and Privacy*, pages 75–86, Los Alamitos, CA, USA, 1984. IEEE Computer Society.

[Gra91] James W. Gray, III. Toward a mathematical foundation for information flow security. In *IEEE Symposium on Security and Privacy*, pages 21–35, 1991.

[Gra92] James W. Gray, III. Toward a mathematical foundation for information. *Journal of Computer Security*, 1(3-4):255–294, 1992.

[GRS09] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 351–360, New York, NY, USA, 2009. ACM.

[GS97] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. American Mathematical Society, second revised edition edition, 1997. Available at `http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.h`

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[HPS01] Maritta Heisel, Andreas Pfitzmann, and Thomas Santen. Confidentiality-preserving refinement. In *Proceedings of 14th IEEE Computer Security Foundations Workshop*, pages 295–305. IEEE Press, 2001.

[JÖ1] Jan Jürjens. Secrecy-preserving refinement. In *Proceedings of FME: Formal Merhods for Increasing Software Productivity*, volume 2021 of *LNCS*, pages 135–152. Springer-Verlag, 2001.

[LSV07] N. Lynch, R. Segala, and F. Vaandrager. Observing branching structure through probabilistic contexts. *SIAM Journal on Computing*, 37(4):977–1013, 2007.

[LV95] Nancy Lynch and Frits Vaandrager. Forward and backward simulations Part I: Untimed systems. *Inf. Comput.*, 121(2):214–233, 1995.

[Mal07] Pasquale Malacaria. Assessing security threats of looping constructs. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 225–235, New York, NY, USA, 2007. ACM.

[Man01] H. Mantel. Preserving information flow properties under refinement. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Press, 2001.

[McS09] Frank McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *SIGMOD '09: Proceedings of the 2009 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2009. ACM.

[ME07] Stephen McCamant and Michael D. Ernst. A simulation-based proof technique for dynamic information flow. In *PLAS '07: Proceedings of the 2007 workshop on Programming languages and analysis for security*, pages 41–46, New York, NY, USA, 2007. ACM.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[MPRV09] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Advances in Cryptology – CRYPTO 2009*, 2009.

[MT07] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society.

[NRS07] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84, New York, NY, USA, 2007. ACM.

[NS08] James Newsome and Dawn Song. Influence: A quantitative approach for data integrity. Technical Report CMU-CyLab-08-005, CyLab, Carnegie Mellon University, 2008.

[PHW04] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non-interference. *J. Comput. Secur.*, 12(1):37–81, 2004.

[PLS00] Anna Philippou, Insup Lee, and Oleg Sokolsky. Weak bisimulation for probabilistic systems. In *CONCUR '00: Proceedings of the 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 334–349, London, UK, 2000. Springer.

[RAW+10] Jason Reed, Adam J. Aviv, Daniel Wagner, Andreas Haeberlen, Benjamin C. Pierce, and Jonathan M. Smith. Differential privacy for collaborative security. In *European Workshop on System Security (EUROSEC)*, April 2010.

[RP10] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, September 2010.

[RRS+10] Indrajit Roy, Hany E. Ramadan, Srinath T.V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for MapReduce. In *Proceedings of the 7th Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[SL95] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2), 1995.

[Smi03] Geoffrey Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 3–13, Pacific Grove, California, 2003.

[SS00] Andrei Sabelfeld and David Sands. Probabilistic non-interefence for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, Cambridge, England, July 2000. IEEE Computer Society Press.

[ST07] Roberto Segala and Andrea Turrini. Approximated computationally bounded simulation relations for probabilistic automata. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 140–156, Venice, Italy, 2007.

A The Truncated Geometric Mechanism

A.1 The Mechanism

The Truncated Geometric Mechanism of Ghosh et al. [GRS09] is an adaptation of the Laplace mechanism made to produce outputs over only a bounded range of discrete values. The Laplace mechanism works by computing the exact result of some statistic f and then adding noise drawn from a Laplace distribution. The amount of noise depends upon both the privacy parameter ϵ and the *sensitivity* of f. The sensitivity of f is the amount the value that f computes can change by adding or removing a single data point from the data set. Formally, the sensitivity of f, denoted $\delta(f)$, is maximum value that $|f(B_1) - f(B_2)|$ can take on where B_1 and B_2 ranges over all pairs of data sets differing by one data point. Using $\kappa_{f,\epsilon}^{\mathsf{LM}}$ to denote the Laplace mechanism applied to the statistic f, we have that $\kappa_{f,\epsilon}^{\mathsf{LM}}(B) = f(B) + \mathsf{Lap}(\delta(f)/\epsilon)$ where $\mathsf{Lap}(b)$ is a random variable producing noise according to the Laplace distribution centered at zero with variance $2b^2$.

To make the Laplace distribution discrete, start by noting that informally the Laplace distribution is two exponential distributions back to back. That is, $\Pr[\mathsf{Lap}(b){=}x] = \Pr[\mathsf{Exponential}(1/b){=}|x|]$ where $\mathsf{Exponential}(\lambda)$ is the exponential distribution with the p.d.f. of $\lambda \exp(-\lambda x)$ at x for $x \leq 0$ and 0 otherwise. Since the discrete version of the exponential distribution is a geometric distribution, one can use two geometric distributions back to back to create a "discrete" Laplace distribution. Formally, $\Pr[\mathsf{Exponential}(\lambda){=}x] = \Pr[\mathsf{Geo}(\exp(-\lambda)){=}\lfloor x \rfloor]$ where $\Pr[\mathsf{Geo}(p){=}k] = p^k(1-p)$ (i.e., p is the "failure probability"). Using DL to denote this distribution, we have that $\Pr[\mathsf{DL}(p){=}n] = p^{|n|}\frac{1-p}{1+p}$.

Next, one must bound the mechanism to produce only results between the minimal and maximum numbers that the computer can represent. For simplicity we assume that the minimum is $-m$ where m is the maximum. Thus, we need that the result of adding noise $f(B) + N$ is such that $-m \leq f(B) + N \leq m$ where N is random variable generating noise. This implies that $-m - f(B) \leq N \leq m - f(B)$ requiring that N depends upon both m and $f(B)$ in addition to ϵ and $\delta(f)$.

At this point, it may be tempting to simply take the discrete Laplace distribution DL and condition on the noise being between $-m - f(B)$ and $m - f(B)$. This will produce a bounded distribution such that the probability of producing two adjacent outputs are within a multiplicative

factor of one another. However, since the condition involves the value of $f(B)$, the distributions resulting from two adjacent data sets may differ. In general, they need not be within a multiplicative factor of one another.

Fixing this problem requires adding extra weight to the probability of producing the extreme results $-m$ and m for $f(B) + N$. Intuitively, this extra weight account for the tails being cut off. Formally, it comes from a system of equations constraining the relationship between each pair of distributions $N(m, f(B_1), \exp(-\epsilon/\delta(f)))$ and $N(m, f(B_2), \exp(-\epsilon/\delta(f)))$ where B_1 and B_2 differ by one data point. Formally,

$$\Pr[N(m,t,p){=}n] = \begin{cases} p^{|n|} * \frac{1}{1+p} & |t+n| = m \\ p^{|n|} * \frac{1-p}{1+p} & -m < t+n < m \\ 0 & \text{otherwise} \end{cases}$$

N produces noise for $\kappa_{f,\epsilon}$, a differentially private mechanism for the statistic f:

$$\kappa_{f,\epsilon}(B) = f(B) + N(m, f(B), \exp(-\epsilon/\delta(f)))$$

Proposition 1 (Differential Privacy)**.** *For all integers $m > 0$, for all functions f from data sets to $\{-m, \ldots, m\}$, the function $\kappa_{f,\epsilon}$ has ϵ-differential privacy.*

Proof. By a lemma similar to Proposition 12, since $\kappa_{f,\epsilon}$ is discrete, it gives ϵ-differential privacy iff for all data sets B_1 and B_2 differing on at most one element, and for all $r \in \mathsf{range}(\kappa_{\mathrm{f},\epsilon})$,

$$\Pr[\kappa_{f,\epsilon}(B_1) = r] \leq \exp(\epsilon) * \Pr[\kappa_{f,\epsilon}(B_2) = r]$$

Note

$$\begin{aligned}
&\Pr[\kappa_{f,\epsilon}(B) = r] \\
&\quad = \Pr[f(B) + N(m, f(B), \exp(-\epsilon/\delta(f))) = r] \\
&\quad = \Pr[N(m, f(B), \exp(-\epsilon/\delta(f))) = r - f(B)] \\
&\quad = \begin{cases} \exp(-\epsilon/\delta(f))^{|r-f(B)|} * \frac{1}{1+\exp(-\epsilon/\delta(f))} & |f(B) + (r - f(B))| = m \\ \exp(-\epsilon/\delta(f))^{|r-f(B)|} * \frac{1-\exp(-\epsilon/\delta(f))}{1+\exp(-\epsilon/\delta(f))} & -m < f(B) + (r - f(B)) < m \\ 0 & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \exp(-|r - f(B)|\epsilon/\delta(f)) * \frac{1}{1+\exp(-\epsilon/\delta(f))} & |r| = m \\ \exp(-|r - f(B)|\epsilon/\delta(f)) * \frac{1-\exp(\epsilon/\delta(f))}{1+\exp(-\epsilon/\delta(f))} & -m < r < m \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

Thus, if $r > m$ or $r < -m$, $\Pr[\kappa_{f,\epsilon}(B_1) = r] = 0 \leq 0 = \exp(-\epsilon) * \Pr[\kappa_{f,\epsilon}(B_2) = r]$. Otherwise, since the normalization factor, which depends on whether $|r| = m$ or not, is the same on each side of the inequality $\Pr[\kappa_{f,\epsilon}(B_1) = r] \leq \exp(\epsilon) * \Pr[\kappa_{f,\epsilon}(B_2) = r]$, the inequality holds iff

$$e{-}|r - f(B_1)|\epsilon/\delta(f) \leq \exp(\epsilon) \exp(-|r - f(B_2)|\epsilon/\delta(f))$$

Since B_1 and B_2 only differ by at most one data point, we know that $|f(B_1) - f(B_2)| \leq \delta(f)$.

Case: $f(B_2) \leq f(B_1)$. In this case, $f(B_1) - f(B_2) \leq \delta(f)$. Let $f(B_1) - f(B_2) = \partial$ so that $\exp(-|r - f(B_1)|\epsilon/\delta(f)) = \exp(-|r - (f(B_2) + \partial)|\epsilon/\delta(f))$.

- Subcase: $|r - f(B_2)| \leq |r - (f(B_2) + \partial)|$. In this case, $\exp(-|r - (f(B_2) + \partial)|\epsilon/\delta(f)) \leq \exp(-|r - f(B_2)|\epsilon/\delta(f))$. Thus,

$$\exp(-|r - f(B_1)|\epsilon/\delta(f)) \leq \exp(\epsilon)\exp(-|r - f(B_2)|\epsilon/\delta(f))$$

since $\epsilon \leq 0$.

- Subcase: $|r - f(B_2)| \geq |r - (f(B_2) + \partial)|$. Let $\partial' = |r - f(B_2)| - |r - (f(B_2) + \partial)|$. Since $\partial' \leq \partial$,

$$\begin{aligned}
\exp(-|r - f(B_1)|\epsilon/\delta(f)) &= \exp(-(|r - (f(B_2)| - \partial')\epsilon/\delta(f)) \\
&= \exp((\partial' - |r - f(B_2)|)\epsilon/\delta(f)) \\
&\leq \exp((\partial - |r - f(B_2)|)\epsilon/\delta(f)) \\
&\leq \exp((\delta(f) - |r - f(B_2)|)\epsilon/\delta(f)) \\
&= \exp((\delta(f)\epsilon/\delta(f)) - (|r - f(B_2)|\epsilon/\delta(f))) \\
&= \exp(\epsilon - (|r - f(B_2)|\epsilon/\delta(f))) \\
&= \exp(\epsilon)\exp(-|r - f(B_2)|\epsilon/\delta(f))
\end{aligned}$$

Case: $f(B_1) \leq f(B_2)$. In this case, $-(f(B_1) - f(B_2)) = f(B_2) - f(B_1) \leq \delta(f)$ Let $f(B_2) - f(B_1) = \partial$ so that

$$\exp(-|r - f(B_2)|\epsilon/\delta(f)) = \exp(-|r - (f(B_1) + \partial)|\epsilon/\delta(f))$$

- Subcase: $|r - f(B_1)| \leq |r - (f(B_1) + \partial)|$. Let $\partial' = |r - (f(B_1) + \partial)| - |r - f(B_1)|$. Since $\partial' \leq \partial$,

$$\begin{aligned}
\exp(-|r - f(B_1)|\epsilon/\delta(f)) &= \exp((-|r - f(B_1)|\epsilon - \delta(f)\epsilon + \delta(f)\epsilon)/\delta(f)) \\
&= \exp(((-|r - f(B_1)| - \delta(f))\epsilon + \epsilon\delta(f))/\delta(f)) \\
&= \exp((-|r - f(B_1)| - \delta(f))\epsilon/\delta(f) + \epsilon\delta(f)/\delta(f)) \\
&= \exp((-|r - f(B_1)| - \delta(f))\epsilon/\delta(f) + \epsilon) \\
&\leq \exp(\epsilon)\exp((-|r - f(B_1)| - \delta(f))\epsilon/\delta(f)) \\
&\leq \exp(\epsilon)\exp((-|r - f(B_1)| - \partial')\epsilon/\delta(f)) \\
&= \exp(\epsilon)\exp(-(|r - f(B_1)| + \partial')\epsilon/\delta(f)) \\
&= \exp(\epsilon)\exp(-|r - (f(B_1) + \partial)|\epsilon/\delta(f)) \\
&= \exp(\epsilon)\exp(-|r - f(B_2)|\epsilon/\delta(f))
\end{aligned}$$

- Subcase: $|r - f(B_1)| \geq |r - (f(B_1) + \partial)|$. In this case, we have that $\exp(-|r - (f(B_1))|\epsilon/\delta(f)) \leq \exp(-|r - f(B_1) + \partial|\epsilon/\delta(f))$. Thus,

$$\exp(-|r - f(B_1)|\epsilon/\delta(f)) \leq \exp(\epsilon)\exp(-|r - f(B_2)|\epsilon/\delta(f))$$

since $\epsilon \leq 0$.

□

The probability of $\kappa_{f,\epsilon}(B)$ being b or more away from $f(B)$ decreases exponentially in b.

Proposition 2 (Utility)**.** $\Pr[|\kappa_{f,\epsilon}(B) - f(B)| \geq b] \leq \frac{2p^b}{1+p}$.

Proof.

$$\begin{aligned}
&\Pr[|\kappa_{f,\epsilon}(B) - f(B)| \geq b] \\
&\quad = 1 - \Pr[-b < \kappa_{f,\epsilon}(B) - f(B) < b] \\
&\quad = 1 - \Pr[-b+1 \leq \kappa_{f,\epsilon}(B) - f(B) \leq b-1] \\
&\quad = 1 - \Pr[-b+1 \leq f(B) + N(m, f(B), \exp(-\epsilon/\delta(f))) - f(B) \leq b-1] \\
&\quad = 1 - \Pr[-b+1 \leq N(m, f(B), \exp(-\epsilon/\delta(f))) \leq b-1] \\
&\quad = 1 - \sum_{n=-b+1}^{b-1} \Pr[N(m, f(B), \exp(-\epsilon/\delta(f))) = n]
\end{aligned}$$

If $b - 1 \geq m - f(B)$ and $-b + 1 \leq -m - t$, then this is $1 - 1 = 0$. If $b - 1 < m - f(B)$ and $-b + 1 > -m - f(B)$, then this is

$$\begin{aligned}
1 - \sum_{n=-b+1}^{b-1} \exp(-\epsilon/\delta(f))^n \frac{1 - \exp(-\epsilon/\delta(f))}{1 + \exp(-\epsilon/\delta(f))} \\
&= 1 - \frac{1 + \exp(-\epsilon/\delta(f)) - 2\exp(-\epsilon/\delta(f))^b}{1 + \exp(-\epsilon/\delta(f))} \\
&= \frac{2p^b}{1+p}
\end{aligned}$$

If $b - 1 < m - f(B)$ and $-b + 1 \leq -m - t$, then this is

$$\begin{aligned}
1 - p^{|-m-t|}\frac{1}{1+p} + \sum_{n=-m-t+1}^{b-1} \exp(-\epsilon/\delta(f))^n \frac{1 - \exp(-\epsilon/\delta(f))}{1 + \exp(-\epsilon/\delta(f))} \\
&= 1 - \frac{1 + \exp(-\epsilon/\delta(f)) - \exp(-\epsilon/\delta(f))^b}{1 + \exp(-\epsilon/\delta(f))} \\
&= \frac{p^b}{1+p}
\end{aligned}$$

If $b \geq m - f(B)$ and $-b > -m - t$, then this is

$$\begin{aligned}
1 - p^{|m-t|}\frac{1}{1+p} + \sum_{n=-b+1}^{m-t-1} \exp(-\epsilon/\delta(f))^n \frac{1 - \exp(-\epsilon/\delta(f))}{1 + \exp(-\epsilon/\delta(f))} \\
&= 1 - \frac{1 + \exp(-\epsilon/\delta(f)) - \exp(-\epsilon/\delta(f))^b}{1 + \exp(-\epsilon/\delta(f))} \\
&= \frac{p^b}{1+p}
\end{aligned}$$

completing the proof. □

A.2 An Implementation

Below is an efficient algorithm for sampling from $N(\mathtt{m}, \mathtt{t}, \mathtt{p})$ for $\mathtt{m} > 0$, $-\mathtt{m} \leq \mathtt{t} \leq \mathtt{m}$, and $0 \leq \mathtt{p} < 1$:

```
01 sample_N(m,t,p)
02   if(flip(p/(1+p)))
03      if(flip(p^(m+t-1)))
04         return(-m-t);
05      else
06         q := (p-1)/(p^(m+t)-p);
07         for(n:=-1; n>-m-t+1; n--)
08            if(flip(q))
09               return(n);
10            q := p*q/(1-q);
11         return(-m-t+1);
12   else
13      if(flip(p^(m-t)))
14         return(m-t);
15      else
16         q := (p-1)*p^t/(p^m-p^t);
17         for(n:=0; n<m-t-1; n++)
18            if(flip(q))
19               return(n);
20            q := p*q/(1-q);
21         return(m-t-1);
```

Each `flip` command uses an independent Bernoulli distribution to select either true or false. `flip`(p) returns true with probability p.

Proposition 3 (Correctness)**.** `sample_N` *samples from* $N(m, t, p)$.

Proof. Let q_n denote the value that variable `q` has the beginning of the nth iteration of the last `for` loop: $q_0 = \frac{(p-1)p^t}{p^m - p^t}$ and $q_n = p * q_{n-1}/(1 - q_{n-1})$ for $0 < n < m - t - 1$. We show by induction over n, that for n between 0 and $m - t - 2$,

$$q_n = \left(1 - \frac{p}{1+p}\right)^{-1} (1 - p^{m-t})^{-1} \prod_{j=0}^{n-1} (1 - q_j)^{-1} p^n \frac{1-p}{1+p}$$

For the base case with $n = 0$,

$$\begin{aligned}
q_0 &= \frac{(p-1)p^t}{p^m - p^t} \\
&= \left(1 - \frac{p}{1+p}\right)^{-1} (1 - p^{m-t})^{-1} \frac{1-p}{1+p} \\
&= \left(1 - \frac{p}{1+p}\right)^{-1} (1 - p^{m-t})^{-1} \prod_{j=0}^{-1} (1 - q_j)^{-1} p^0 \frac{1-p}{1+p} \\
&= \left(1 - \frac{p}{1+p}\right)^{-1} (1 - p^{m-t})^{-1} \prod_{j=0}^{n-1} (1 - q_j)^{-1} p^n \frac{1-p}{1+p}
\end{aligned}$$

For the inductive case, assume this is true for $n-1$. Then,

$$\begin{aligned}
q_n &= p * q_{n-1}/(1-q_{n-1}) \\
&= p * \left(\left(1 - \frac{p}{1+p}\right)^{-1} (1-p^{m-t})^{-1} \prod_{j=0}^{(n-1)-1} (1-q_j)^{-1} p^{(n-1)} \frac{1-p}{1+p} \right) * (1-q_{n-1})^{-1} \\
&= \left(1 - \frac{p}{1+p}\right)^{-1} (1-p^{m-t})^{-1} \prod_{j=0}^{n-1} (1-q_j)^{-1} p^n \frac{1-p}{1+p}
\end{aligned}$$

If the last `for` loop executes, then with probability $\prod_{j=0}^{n-1}(1-q_j)q_n$ it will stop at the nth iteration and return n for values of n between 0 to $m-t-2$ (inclusive). Using above equation,

$$\begin{aligned}
&\prod_{j=0}^{n-1}(1-q_j)q_n \\
&\quad = \left(\prod_{j=0}^{n-1}(1-q_j)\right) \left(\left(1 - \frac{p}{1+p}\right)^{-1} (1-p^{m-t})^{-1} \prod_{j=0}^{n-1} (1-q_j)^{-1} p^n \frac{1-p}{1+p} \right) \\
&\quad = \left(1 - \frac{p}{1+p}\right)^{-1} (1-p^{m-t})^{-1} p^n \frac{1-p}{1+p}
\end{aligned}$$

Since the probability of the `for` loop executing is $\left(1 - \frac{p}{1+p}\right)(1-p^{m-t})$, this implies that the probability of returning n such that $0 \le n \le m-t-2$ is $p^n \frac{1-p}{1+p} = p^{|n|}\frac{1-p}{1+p}$.

For $0 \le n = m-t$, the probability of returning $m-t$ is $(1-\frac{p}{1+p})p^{m-t} = p^{m-t} * \frac{1}{1+p} = p^{|m-t|} * \frac{1}{1+p}$.

The probability of the `for` running until completion and returning $m-t-1$ is equal to the probability that none of the other values of n is returned. That is, the probability `flip(p/(1+p))` returning false less the probability of some other number between 0 and $m-t$ being returned:

$$(1 - \frac{p}{1+p}) - p^{m-t} * \frac{1}{1+p} - \sum_{n=0}^{m-t-2} p^j(1-p)/(1+p) = p^{m-t-1}\frac{1-p}{1+p} = p^{|m-t-1|}\frac{1-p}{1+p}$$

Nearly the same reasoning shows that the negative values for noise also have the correct probabilities. □

Assuming that all the operations in `sample_N` including `flip` are constant time, `sample_N` runs in expected constant time.

Proposition 4 (Runtime Complexity)**.** `sample_N` *runs in* $O(1)$ *expected time.*

Proof. The expected running time is $\sum_{n=-m-t}^{m-t} \Pr[N(m,t,p)=n] * T_n$ where T_n is the running time of `sample_N` when it produces n. The running time is constant in the case where `sample_N` produces $m-t$ or $-m-t$. The running time T_n is $O(|n|)$ for n such that $-m-t < n < m-t$. Thus,

ignoring constants, the expected running time is

$$\sum_{n=-m-t+1}^{m-t-1} p^{|n|}\frac{1-p}{1+p}n \leq \frac{1-p}{1+p} * 2 * \sum_{n=0} \max(|-m-t+1|, m-t-1)p^n n \tag{1}$$

$$\leq \frac{1-p}{1+p} * 2 * \sum_{n=0} \infty p^n n \tag{2}$$

$$= \frac{1-p}{1+p} * 2 * \frac{p}{1-p} \tag{3}$$

$$= \frac{2p}{1+p} \tag{4}$$

$$\leq 2 \tag{5}$$

where line 3 follows from the expected value of the geometric distribution. (Recall that we are using p to denote the failure probability unlike most references, which use $1-p$ for the failure probability.) Thus, it is expected to run in constant time. □

A.3 Using the Mechanism for the Sanitization Functions COUNT and SUM

We use the above privacy mechanism to implement sanitization functions similar to the ones that PINQ provides. Due to space constraints, we focus on two representative ones: COUNT and SUM. Since we use a bounded discrete privacy mechanism over integers, our implementations differ from the implementations found in PINQ. We force data points to be integers between -100 and 100 whereas PINQ bounds the sensitivity of functions by mapping data points to doubles between -1 and 1. (Our range may be made larger without affecting our results.) We then use numbers outside this range to encode objects other than data points such as queries.

Given these, we implement our PINQ-like system as follows. `datapoint(y)` on line 09 of the code of Figure 1 would be implemented as a function with body `return(-100 <= y && y <= 100)`. `emptyArray(`x`)` must be implemented to store a value outside of $\{-100, \ldots, 100\}$ so that data points can be distinguished from empty spots. The program uses numbers larger than 100 to indicate queries: 101 denotes COUNT and 102 denotes SUM. Given 101 or 102, `get_sanitization_funct(y)` returns a function that computes the count statistic or sum statistic, respectively. COUNT is computed with

```
01 count(dPts)
02   count := 0;
03   for(j:=0; j<t; j++)
04     for(k:=0; k<maxPts; k++)
05       if(-100 <= dp[j][k] <= 100)
06         count++;
07       else
08         break;
09   s := t*maxPts/2;
10   noise:=s+sample_N(s,count-s,exp(-e/1));
11   result:=count+noise;
12   return(result);
```

and SUM with

```
01 sum(dPts)
02   sum := 0;
03   for(j:=0; j<t; j++)
04     for(k:=0; k<maxPts; k++)
05       if(-100 <= dp[j][k] <= 100)
06           sum := sum+dp[j][k];
07         else
08           break;
09   noise :=
       sample_N(t*maxPts*100,sum,exp(-e/100));
10   result := sum+noise;
11   return(result);
```

where `sample_N` is as defined above and `e` stores the value for the privacy bound ϵ. We add and subtract `t*maxPts` in the calculation of the noise in COUNT to shift the noise over to keep the value count positive.

B Automaton Model

B.1 Probability of Action Sequences

We use $\langle L, s\rangle(\vec{i})(\vec{a}, s')$ to denote the probability of the automaton (starting in state s) producing the trace $\vec{a}$ and ending in the state s' after producing the last action of $\vec{a}$ given that the available inputs are $\vec{i}$. $\langle L, s\rangle(\vec{i})(\vec{a}, s')$ is defined as follows:

$$\begin{aligned}
\langle L, s\rangle(i{:}\vec{i})(i{:}\vec{a}, s') &= \sum_{s''\in S} \mu(s'')\langle L, s''\rangle(\vec{i})(\vec{a}, s') && \text{if } s \xrightarrow{i} \mu \\
\langle L, s\rangle(\vec{i})(o{:}\vec{a}, s') &= \sum_{s''\in S} \mu(s'')\langle L, s''\rangle(\vec{i})(\vec{a}, s') && \text{if } s \xrightarrow{o} \mu \\
\langle L, s\rangle(\vec{i})([\,], s) &= 1 \\
\langle L, s\rangle(\vec{i})(\vec{a}, s') &= 0 && \text{otherwise}
\end{aligned}$$

where $i \in I$ and $o \in O$. The first line in the above definition, for example, considers the case where the state s transitions to a new state under the input i according to the distribution μ. It states the probability of starting in the state s, consuming the input i, and then performing the actions $\vec{a}$ ending in state s' given that $\vec{i}$ remain available inputs. This probability is the sum of the probabilities of transitioning to a state s'' and then performing the actions $\vec{a}$ from s'', ending in state s' given that $\vec{i}$ are available inputs.

Proposition 5. *For all automata $\langle L, s\rangle$, $\vec{a}$ in A^*, s' in S, and $\vec{i}$ in I^*, $\langle L, s\rangle(\vec{i})(\vec{a}, s')$ is well defined and between 0 and 1.*

Proof. Proof by induction over the structure of $\vec{a}$.

Case: $\vec{a} = [\,]$. $\langle L, s\rangle(\vec{i})(\vec{a}, s')$ is 1 if $s' = s$, and $\langle L, s\rangle(\vec{i})(\vec{a}, s')$ is 0 for $s' \neq s$.

Case: $\vec{a} = i{:}\vec{a}'$. If there does not exist $\vec{i}'$ such that $\vec{i} = i{:}\vec{i}'$ and $s \xrightarrow{i} \mu$, then $\langle L, s\rangle(\vec{i})(\vec{a}, s') = 0$. If there does exist such a $\vec{i}'$, then $\langle L, s\rangle(\vec{i})(\vec{a}, s') = \sum_{s'' \in S} \mu(s'')\langle L, s''\rangle(\vec{i}')(\vec{a}')$. By the inductive hypothesis, $\langle L, s''\rangle(\vec{i}')(\vec{a}', s')$ is well defined and between 0 and 1 for all s''. Since μ is a distribution over states and the events of being in a state are mutually exclusive, $\sum_{s'' \in S} \mu(s'') = 1$. Let $s_{\mathsf{max}} = \arg\max_{s'' \in S} \langle L, s''\rangle(\vec{i}')(\vec{a}', s')$.

$$\begin{aligned}
[\![\langle L, s\rangle]\!](\vec{i})(\vec{a}, s') &= \sum_{s'' \in S} \mu(s'')\langle L, s'\rangle(\vec{i}')(\vec{a}', s') \\
&\leq \sum_{s'' \in S} \mu(s'')\langle L, s_{\mathsf{max}}\rangle(\vec{i}')(\vec{a}', s') \\
&= \langle L, s_{\mathsf{max}}\rangle(\vec{i}')(\vec{a}', s') \\
&\leq 1
\end{aligned}$$

Case: $\vec{a} = o{:}\vec{a}'$. If there does not exist μ such that $s \xrightarrow{o} \mu$, then $\langle L, s\rangle(\vec{i})(\vec{a}, s') = 0$. If there does, then $\langle L, s\rangle(\vec{i})(\vec{a}, s') = \sum_{s'' \in S} \mu(s'')\langle L, s''\rangle(\vec{i})(\vec{a}', s')$ and we can use the inductive hypothesis as above. □

A helpful proposition about our model follows.

Proposition 6. *For all* PLTS L*, states* $s', s'' \in S$*,* $\vec{i}' \in I^*$*, and* $\vec{h} \in H^*$*,*

$$\langle L, s''\rangle(\vec{i}')(\vec{h}{:}\vec{a}', s') = \sum_{s''' \in S} \langle L, s''\rangle([\,])(\vec{h}, s''') * \langle L, s'''\rangle(\vec{i}')(\vec{a}', s')$$

Proof. Proof by induction over the structure of $\vec{h}$. In the case where $\vec{h} = [\,]$, $\langle L, s''\rangle([\,])(\vec{h}, s''') = 1$ when $s''' = s''$ and 0 otherwise. Thus,

$$\begin{aligned}
\sum_{s''' \in S} \langle L, s''\rangle([\,])([\,], s''') * \langle L, s'''\rangle(\vec{i}')(\vec{a}', s') &= 1 * \langle L, s''\rangle([\,]{:}\vec{i}')(\vec{a}', s') \\
&= \langle L, s''\rangle(\vec{i}')(\vec{h}{:}\vec{a}', s')
\end{aligned}$$

Case: $\vec{h} = h{:}\vec{h}'$ for some h and $\vec{h}'$. If $s'' \xrightarrow{h} \mu'$, then

$$\begin{aligned}
&\sum_{s''' \in S} \langle L, s''\rangle([\,])(h{:}\vec{h}', s''') * \langle L, s'''\rangle(\vec{i}')(\vec{a}', s') \\
&\quad= \sum_{s''' \in S} \left(\sum_{s'''' \in S} \mu'(s'''')\langle L, s''''\rangle([\,])(\vec{h}', s''') \right) * \langle L, s'''\rangle(\vec{i}')(\vec{a}', s') \\
&\quad= \sum_{s'''' \in S} \mu'(s'''') \sum_{s''' \in S} \langle L, s''''\rangle([\,])(\vec{h}', s''') * \langle L, s'''\rangle(\vec{i}')(\vec{a}', s') \\
&\quad= \sum_{s'''' \in S} \mu'(s'''')\langle L, s''''\rangle(\vec{i}')(\vec{h}'{:}\vec{a}', s') \\
&\quad= \langle L, s''\rangle(\vec{i}')(h{:}\vec{h}'{:}\vec{a}', s')
\end{aligned}$$

where third line follows from the inductive hypothesis. If for no μ', then $s'' \xrightarrow{h} \mu'$, $\langle L, s''\rangle(\vec{i}')(\vec{h}{:}\vec{a}', s') = 0 = \sum_{s''' \in S} \langle L, s''\rangle([\,])(\vec{h}, s''') * \langle L, s'''\rangle(\vec{i}')(\vec{a}', s')$ since $\langle L, s''\rangle([\,])(\vec{h}, s''') = 0$ for all $s''' \in S$. □

B.2 Extended Transitions

We define $s \stackrel{a}{\Rightarrow} \nu$ so that $\nu(s')$ is the probability of reaching the H-disabled state s' from the state s where a is the action performed from state s:

$$\nu(s') = \sum_{s'' \in S} \mu(s'') \sum_{\vec{h} \in H^*} \langle L, s'' \rangle([\,])(\vec{h}, s') \qquad\qquad s' \text{ is } H\text{-disabled}$$

and $\nu(s') = 0$ otherwise where $s \stackrel{a}{\rightarrow} \mu$. Thus, the probability of reaching the H-disabled state s' from s by performing the action a followed by a sequence of hidden actions $\vec{h}$ is calculated by considering each s'' that is reachable by performing the single action a from s. For each such s'' we multiply the probability of ending up in s'' by performing an a from s with the the probability of reaching s' from s'' by performing a sequence of hidden actions (the inner sum). The value $\nu(s')$ is then calculated by adding the probabilities corresponding to each s''. Since all $\vec{h}$ in H^* contain only actions from H, an execution with the action sequence $\vec{h}$ cannot leave an H-disabled state. Thus, $\nu(s')$ is the probability of s' being the first H-disabled state reached. If there is no μ such that $s \stackrel{a}{\rightarrow} \mu$, then there is no ν such that $s \stackrel{a}{\Rightarrow} \nu$.

For notational convenience we extend the transition relation $\rightarrow$ to $S_\perp$ by having no transitions to nor from $\perp$. This implies that

$$\begin{aligned}
&\langle L, \perp \rangle(\vec{i})([\,], \perp) = 1 \\
&\langle L, s \rangle(\vec{i})(\vec{a}, \perp) = 1 - \sum_{s' \in S} \langle L, s \rangle(\vec{i})(\vec{a}, s) \\
&\langle L, \perp \rangle(\vec{i})(\vec{a}, x) = 0 \qquad\qquad \text{if } \vec{a} \neq [\,] \text{ or } x \neq \perp
\end{aligned}$$

Thus, $\Pr[\,\lfloor [\![\langle L, \perp \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,]$ is 1 if $\vec{e} = [\,]$ and 0 otherwise, which matches the intuition that a nonterminating program which never interacts with the data examiner will only have the empty trace as a prefix.

Proposition 7. *For all states s and actions a, $s \stackrel{a}{\Rightarrow} \nu$ implies that ν is a distribution over $S_\perp$.*

Proof. To prove that ν is a distribution over $S_\perp$, we must show that for all $x \in S_\perp$, $0 \leq \nu(x) \leq 1$ and $\sum_{x \in S_\perp} \nu(x) = 1$. We start by proving that $\sum_{x \in S'} \nu(x) \leq 1$ by introducing a function η.

Given the set S' of H-disabled states, let η be defined as follows:

$$\begin{aligned}
&\eta(n, s) = 1 && s \in S' \\
&\eta(n, s) = 0 && \text{when } n = 0 \text{ and } s \notin S' \\
&\eta(n, s) = \sum_{h \in H} \sum_{s'' \in S} \mu_h(s'') \eta(n - 1, s'') && \text{otherwise}
\end{aligned}$$

where $s \stackrel{h}{\rightarrow} \mu_h$ and n is a natural number.

Proof by induction over n shows that $\eta(n, s) = \sum_{\vec{h} \in H^{\leq n}} \sum_{s' \in S'} \langle L, s \rangle([\,])(\vec{h}, s')$. where $H^{\leq n} = H^n \cup H^{\leq n-1}$ for $n \leq 1$ and $H^{\leq 0} = H^0 = \{[\,]\}$. In the base case, $n = 0$, if $s \in S'$, then $\eta(n, s) = 1 = \sum_{s' \in S'} \langle L, s \rangle([\,])([\,], s')$ since $\langle L, s \rangle([\,])([\,], s) = 1$, $\langle L, s \rangle([\,])([\,], s') = 0$ for $s \neq s'$, and $s \in S'$. If $s \notin S'$, $\eta(n, s) = 0 = \sum_{s' \in S'} \langle L, s \rangle([\,])([\,], s')$ since $\langle L, s \rangle([\,])([\,], s') = 0$ for $s \neq s'$ and $s \notin S'$ whereas $s' \in S'$.

In the inductive case, if $s \in S'$, then $\langle L, s\rangle([\,])(\vec{h}, s') = 0$ if $\vec{h} \neq [\,]$ or $s' \neq s$ since s is H-disabled. Thus, $\eta(n+1, s) = 1 = \sum_{\vec{h} \in H^{\leq n+1}} \sum_{s' \in S'} \langle L, s\rangle([\,])(\vec{h}, s')$ since $\langle L, s\rangle([\,])([\,], s) = 1$. If $s \notin S'$, then

$$\begin{aligned}
\eta(n+1, s) &= \sum_{h \in H} \sum_{s'' \in S} \mu_h(s'')\eta(n, s'') && (6)\\
&= \sum_{h \in H} \sum_{s'' \in S} \mu_h(s'') \sum_{\vec{h} \in H^{\leq n}} \sum_{s' \in S'} \langle L, s\rangle([\,])(\vec{h}, s') && (7)\\
&= \sum_{s' \in S'} \sum_{h \in H} \sum_{\vec{h} \in H^{\leq n}} \sum_{s'' \in S} \mu_h(s'')\langle L, s\rangle([\,])(\vec{h}, s') && (8)\\
&= \sum_{s' \in S'} \sum_{h \in H} \sum_{\vec{h} \in H^{\leq n}} \langle L, s\rangle([\,])(h{:}\vec{h}, s') && (9)\\
&= \sum_{s' \in S'} \langle \sum_{h \in H} \sum_{\vec{h} \in H^{\leq n}} \langle L, s\rangle([\,])(h{:}\vec{h}, s')\rangle) + \langle L, s\rangle([\,])([\,], s') && (10)\\
&= \sum_{s' \in S'} \sum_{\vec{h} \in H^{\leq n+1}} \langle L, s\rangle([\,])(\vec{h}, s') && (11)\\
&= \sum_{\vec{h} \in H^{\leq n+1}} \sum_{s' \in S'} \langle L, s\rangle([\,])(\vec{h}, s') && (12)
\end{aligned}$$

Line 7 follows from the inductive hypothesis. Line 9 follows since s is H-enabled. Line 10 follows from $\langle L, s\rangle([\,])([\,], s') = 0$ since $s \notin S'$.

Induction over n can also show that $0 \leq \eta(n, s) \leq 1$ since μ_h is always a distribution.

We use η to show the following:

$$\begin{aligned}
\sum_{s' \in S'} \nu(s') &= \sum_{s' \in S'} \sum_{s'' \in S} \mu(s'') \sum_{\vec{h} \in H^*} \langle L, s''\rangle([\,])(\vec{h}, s')\\
&= \sum_{s'' \in S} \mu(s'') \sum_{\vec{h} \in H^*} \sum_{s' \in S'} \langle L, s''\rangle([\,])(\vec{h}, s')\\
&= \sum_{s'' \in S} \mu(s'') \lim_{n \to \infty} \sum_{\vec{h} \in H^{\leq n}} \sum_{s' \in S'} \langle L, s''\rangle([\,])(\vec{h}, s')\\
&= \sum_{s'' \in S} \mu(s'') \lim_{n \to \infty} \eta(n, s'')\\
&\leq \sum_{s'' \in S} \mu(s'')1\\
&\leq 1
\end{aligned}$$

where $s \xrightarrow{a} \mu$.

For all $s' \in S$, if s' is H-enabled, $\nu(s') = 0$. Thus, $\sum_{s \in S} \nu(s) = \sum_{s' \in S'} \nu(s') \leq 1$. Furthermore, for all $s \in S$, $0 \leq \nu(s)$ and $0 \leq \sum_{s \in S} \nu(s)$ since no operations that could introduce negative numbers is every used in computing $\nu(s)$. Since $\nu(\perp) = 1 - \sum_{s \in S} \nu(s)$, $0 \leq \nu(\perp) \leq 1$ and $\sum_{x \in S_\perp} \nu(x) = 1$. Since for all s, $0 \leq \nu(s)$ and $\sum_{s \in S} \nu(s) \leq 1$, it must be the case that $\nu(s) \leq 1$. □

Given such an automaton $M = \langle L, s\rangle$, we define $[\![M]\!]$ to be a function from input sequences to

a distribution over trace prefixes (finite action sequences).

$$\Pr[\,[\![M]\!](\vec{i}) \sqsupseteq \vec{a}\,] = \sum_{s' \in S} M(\vec{i})(\vec{a}, s')$$

We write $\lfloor \vec{a} \rfloor_E$ for restricting the action sequence $\vec{a}$ to some subset E of A. Formally, $\lfloor [\,] \rfloor_E = [\,]$, $\lfloor a{:}\vec{a} \rfloor_E = a{:}\lfloor \vec{a} \rfloor_E$ if $a \in E$, and $\lfloor a{:}\vec{a} \rfloor_E = \lfloor \vec{a} \rfloor_E$, otherwise. For infinite sequences $\vec{a}$ with only a finite number of elements from E, $\lfloor \vec{a} \rfloor_E$ is the finite sequence that results from $\lfloor \vec{a}' \rfloor_E$ where $\vec{a}'$ is the finite prefix of $\vec{a}$ holding all the elements from E. If $\vec{a}$ contains an infinite number of elements from E, then $\lfloor \vec{a} \rfloor_E$ is the infinite sequence whose jth entry is the jth element of E in $\vec{a}$.

Given an automaton M, $\Pr[\lfloor [\![M]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,]$ is the probability of the data examiner seeing $\vec{e} \in E^*$ as a prefix given that the available inputs are $\vec{i}$. To calculate $\Pr[\lfloor [\![M]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,]$, consider the set $\gamma(\vec{e})$ of action sequences $\vec{a}$ such that $\lfloor \vec{a} \rfloor_E = \vec{e}$ and ends with the last element of $\vec{e}$. That is, $\gamma(\vec{e}) = \{\, \vec{a} \in A^* \mid \lfloor \vec{a} \rfloor_E = \vec{e} \wedge \mathsf{last}(\vec{a}) = \mathsf{last}(\vec{e}) \,\}$ with the special case that $\gamma([\,]) = \{[\,]\}$. To calculate $\Pr[\lfloor [\![M]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,]$, we need not consider all $\vec{a}$ such that $\lfloor \vec{a} \rfloor_E = \vec{e}$. Rather, we may focus only on those in $\gamma(\vec{e})$ since every $\vec{a}$ such that $\lfloor \vec{a} \rfloor_E = \vec{e}$ will have a prefix in $\gamma(\vec{e})$. Since it is impossible to see two different prefixes from $\gamma(\vec{e})$ during the same execution (no element of $\gamma(\vec{e})$ is the prefix of another), they are mutually exclusive. Thus, $\Pr[\lfloor [\![M]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,] = \sum_{\vec{a} \in \gamma(\vec{e})} \Pr[[\![M]\!](\vec{i}) \sqsupseteq \vec{e}\,]$.

B.3 Some Helpful Propositions

We need some propositions about our model to prove the soundness of unwinding later in Appendix E.

Let $H^*{:}\gamma(\vec{e}')$ stand for $\{\, \vec{a} \in A^* \mid \exists \vec{h} \in H^*, \exists \vec{a}'' \in \gamma(\vec{e}'), \vec{a} = \vec{h}{:}\vec{a}'' \,\}$.

We use $\gamma'(\vec{e})$ do denote those action sequences of $\gamma(\vec{e})$ that do not start with a hidden output from H: $\gamma'(\vec{e}) = \{\, \vec{a} \in \gamma(\vec{e}) \mid \vec{a} = [\,] \vee \exists a \in A - H, \exists \vec{a}' \in A^*, \vec{a} = a{:}\vec{a}' \,\}$ where $A - H$ is the set difference.

Proposition 8. *For all $\vec{e} \in E^*$, if $\vec{e} \neq [\,]$, then $H^*{:}\gamma'(\vec{e}) = \gamma(\vec{e})$.*

Proof. To show that $H^*{:}\gamma'(\vec{e}) \subseteq \gamma(\vec{e})$, note that for all $\vec{a} \in H^*{:}\gamma'(\vec{e})$, there exists $\vec{h} \in H^*$ and $\vec{a}' \in \gamma'(\vec{e}) \subseteq \gamma(\vec{e})$ such that $\vec{a} = \vec{h}{:}\vec{a}'$. Furthermore, $\lfloor \vec{h}{:}\vec{a}' \rfloor_E = \lfloor \vec{a}' \rfloor_E = \vec{e}'$ since $H \cap E = \emptyset$. Since $\vec{e} \neq [\,]$, $\vec{a} \neq [\,]$ and $\mathsf{last}(\vec{h}{:}\vec{a}') = \mathsf{last}(\vec{a}') = \mathsf{last}(\vec{e})$. Thus, $\vec{h}{:}\vec{a}' \in \gamma(\vec{e})$.

To show that $\gamma(\vec{e}') \subseteq H^*{:}\gamma'(\vec{e})$, for any $\vec{a} \in \gamma(\vec{e}')$, either $\vec{a} \in H^*$ or there exists $\vec{h} \in H^*$, $a \in A - H$, and $\vec{a}' \in A^*$ such that $\vec{a} = \vec{h}{:}a{:}\vec{a}'$. The first case cannot arise since it would imply that $\vec{e} = [\,]$ since $\vec{e} = \lfloor \vec{a} \rfloor_E = [\,]$. For the second case, since $\vec{e} = \lfloor \vec{h}{:}a{:}\vec{a}' \rfloor_E = \lfloor a{:}\vec{a}' \rfloor_E$ and $\mathsf{last}(\vec{e}) = \mathsf{last}(\vec{h}{:}a{:}\vec{a}') = \mathsf{last}(a{:}\vec{a}')$. Thus, $a{:}\vec{a}' \in \gamma(\vec{e})$. Thus, $\vec{h}{:}a{:}\vec{a}' \in H^*{:}\gamma'(\vec{e})$. □

Proposition 9.

$$\Pr[\,[\![\langle L,s\rangle]\!](i{:}\vec{i})\sqsupseteq i{:}\vec{a}\,] = \sum_{s'\in S}\mu(s')\Pr[\,[\![\langle L,s'\rangle]\!](\vec{i})\sqsupseteq\vec{a}\,] \qquad \textit{if } s\xrightarrow{i}\mu \textit{ and } i\in I$$

$$\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq o{:}\vec{a}\,] = \sum_{s'\in S}\mu(s')\Pr[\,[\![\langle L,s'\rangle]\!](\vec{i})\sqsupseteq\vec{a}\,] \qquad \textit{if } s\xrightarrow{o}\mu \textit{ and } o\in O$$

$$\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq[\,]\,] = 1$$

$$\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq\vec{a}\,] = 0 \qquad \textit{otherwise}$$

and $0\le\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq\vec{a}\,]\le 1$.

Proof. For the first equation:

$$\begin{aligned}\Pr[\,[\![\langle L,s\rangle]\!](i{:}\vec{i})\sqsupseteq i{:}\vec{a}\,] &= \sum_{s''\in S}\langle L,s\rangle(i{:}\vec{i})(i{:}\vec{a},s'')\\ &= \sum_{s''\in S}\sum_{s'\in S}\mu(s')\langle L,s\rangle(\vec{i})(\vec{a},s'')\\ &= \sum_{s'\in S}\mu(s')\sum_{s'\in S}\langle L,s\rangle(\vec{i})(\vec{a},s'')\\ &= \sum_{s'\in S}\mu(s')\Pr[\,\langle L,s\rangle(\vec{i})\sqsupseteq\vec{a}\,]\end{aligned}$$

For the second equation:

$$\begin{aligned}\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq o{:}\vec{a}\,] &= \sum_{s''\in S}\langle L,s\rangle(\vec{i})(o{:}\vec{a},s'')\\ &= \sum_{s''\in S}\sum_{s'\in S}\mu(s')\langle L,s\rangle(\vec{i})(\vec{a},s'')\\ &= \sum_{s'\in S}\mu(s')\sum_{s'\in S}\langle L,s\rangle(\vec{i})(\vec{a},s'')\\ &= \sum_{s'\in S}\mu(s')\Pr[\,\langle L,s\rangle(\vec{i})\sqsupseteq\vec{a}\,]\end{aligned}$$

For the third equation: $\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq[\,]\,] = \sum_{s'\in S}\langle L,s\rangle(\vec{i})([\,],s') = 1$ since $\langle L,s\rangle(\vec{i})([\,],s) = 1$ and $\langle L,s\rangle(\vec{i})([\,],s') = 0$ for all $s'\neq s$.

For the forth equation: $\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq\vec{a}\,] = \sum_{s'\in S}\langle L,s\rangle(\vec{i})([\,],s') = 0$ since $\langle L,s\rangle(\vec{i})([\,],s') = 0$ for all s'.

To show that $0\le\Pr[\,[\![\langle L,s\rangle]\!](\vec{i})\sqsupseteq\vec{a}\,]\le 1$, we use proof by induction over the structure of $\vec{a}$.

Case: $\vec{a} = [\,]$. $[\![\langle L,s\rangle]\!](\vec{i})(\vec{a})$ is 1.

Case: $\vec{a} = i{:}\vec{a}'$. If there does not exist $\vec{i}'$ such that $\vec{i} = i{:}\vec{i}'$ and $s\xrightarrow{i}\mu$, then $[\![\langle L,s\rangle]\!](\vec{i})(\vec{a}) = 0$. If there does exist such a $\vec{i}'$, then $[\![\langle L,s\rangle]\!](\vec{i})(\vec{a}) = \sum_{s'\in S}\mu(s')[\![\langle L,s'\rangle]\!](\vec{i}')(\vec{a}')$. By the inductive hypothesis, $[\![\langle L,s'\rangle]\!](\vec{i}')(\vec{a}')$ is well defined and between 0 and 1 for all s'. Since μ is a distribution

over states and the events of being in a state are mutually exclusive, $\sum_{s' \in S} \mu(s') = 1$. Let $s_{\mathsf{max}} = \arg\max_{s' \in S} [\![\langle L, s' \rangle]\!](\vec{i}')(\vec{a}')$.

$$\begin{aligned}
[\![\langle L, s \rangle]\!](\vec{i})(\vec{a}) &= \sum_{s' \in S} \mu(s') [\![\langle L, s' \rangle]\!](\vec{i}')(\vec{a}') \leq \sum_{s' \in S} \mu(s') [\![\langle L, s_{\mathsf{max}} \rangle]\!](\vec{i}')(\vec{a}') \\
&= [\![\langle L, s_{\mathsf{max}} \rangle]\!](\vec{i}')(\vec{a}') \\
&\leq 1
\end{aligned}$$

Case: $\vec{a} = o{:}\vec{a}'$. If there does not exist μ such that $s \xrightarrow{o} \mu$, then $[\![\langle L, s \rangle]\!](\vec{i})(\vec{a}) = 0$. If there does, then $[\![\langle L, s \rangle]\!](\vec{i})(\vec{a}) = \sum_{s' \in S} \mu(s') [\![\langle L, s' \rangle]\!](\vec{i})(\vec{a}')$ and we can use the inductive hypothesis as above. □

Proposition 10. *For all H-disabled states s, a in $D \cup Q \cup R$, $\vec{e}$ in E^*, and $\vec{i}$ in I^*, if $\vec{e} \neq [\,]$, $s \xrightarrow{a} \mu$, and $s \stackrel{a}{\Rightarrow} \nu$, then*

$$\sum_{\vec{a} \in \gamma(\vec{e})} \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') * \langle L, s'' \rangle(\vec{i})(\vec{a}, s') = \sum_{x \in S_\perp} \nu(x) \Pr[\, \lfloor [\![\langle L, x \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,]$$

Proof.

$$\sum_{\vec{a} \in \gamma(\vec{e})} \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') * \langle L, s'' \rangle(\vec{i})(\vec{a}, s') \tag{13}$$

$$= \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') \sum_{\vec{a} \in \gamma(\vec{e})} \langle L, s'' \rangle(\vec{i})(\vec{a}, s') \tag{14}$$

$$= \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') \sum_{\vec{a} \in H^* : \gamma'(\vec{e})} \langle L, s'' \rangle(\vec{i})(\vec{a}, s') \tag{15}$$

$$= \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') \sum_{\vec{h} \in H^*} \sum_{\vec{a} \in \gamma'(\vec{e})} \langle L, s'' \rangle(\vec{i})(\vec{h}{:}\vec{a}, s') \tag{16}$$

$$= \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') \sum_{\vec{h} \in H^*} \sum_{\vec{a} \in \gamma(\vec{e})} \sum_{s''' \in S} \langle L, s'' \rangle([\,])(\vec{h}, s''') * \langle L, s''' \rangle(\vec{i})(\vec{a}, s') \tag{17}$$

$$= \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') \sum_{\vec{h} \in H^*} \sum_{\vec{a} \in \gamma(\vec{e})} \sum_{s''' \in S'} \langle L, s'' \rangle([\,])(\vec{h}, s''') * \langle L, s''' \rangle(\vec{i})(\vec{a}, s') \tag{18}$$

$$= \sum_{s''' \in S'} \left(\sum_{s'' \in S} \mu(s'') \sum_{\vec{h} \in H^*} \langle L, s'' \rangle([\,])(\vec{h}, s''') \right) \sum_{\vec{a} \in \gamma(\vec{e})} \sum_{s' \in S} \langle L, s''' \rangle(\vec{i})(\vec{a}, s') \tag{19}$$

$$= \sum_{s''' \in S'} \nu(s''') \sum_{\vec{a} \in \gamma(\vec{e})} \Pr[\, [\![\langle L, s''' \rangle]\!](\vec{i}) \sqsupseteq \vec{a}\,] \tag{20}$$

$$= \sum_{s''' \in S'} \nu(s''') \Pr[\, \lfloor [\![\langle L, s''' \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,] \tag{21}$$

$$= \sum_{x \in S_\perp} \nu(x) \Pr[\, \lfloor [\![\langle L, x \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,] \tag{22}$$

where S' is the subset of states S that are H-disabled. Line 15 follows from Proposition 8. Line 16 follows since there is a one-to-one correspondence between elements of $H^*{:}\gamma'(\vec{e})$ and $H^* \times \gamma'(\vec{e})$ given

as $\vec{a} \in H^*{:}\gamma'(\vec{e})$ corresponding to $\langle \vec{h}, \vec{a} \rangle$ where $\vec{h}$ is the largest sequence of H^* such that $\vec{a} = \vec{h}{:}\vec{a}'$ for some $\vec{a}'$. Line 17 follows from Proposition 6. Line 18 follows since $\vec{a} \in \gamma'(\vec{e})$ starting with an action not in H implies that $\langle L, s''' \rangle(\vec{i})(\vec{a}, s') = 0$ for any state that is H-enabled. Line 22 follows from $\Pr[\, \lfloor [\![\langle L, \bot \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq e{:}\vec{e}\,] = 0$ since $e{:}\vec{e} \neq [\,]$ and $\nu(s''') = 0$ for any H-enabled state s'''. □

Informally speaking the following proposition shows how we can account for transitions on hidden actions in calculating the probability of observing a particular behavior from a given state. The first part of the proposition states that the probability of observing the sequence $\vec{e}$ starting from the state s given the input sequence $d{:}\vec{i}'$ can be calculated by considering those states that are reachable from s by performing the action d followed by a sequence of hidden actions. For each such reachable state we take the probability of being in that state and multiply it with the probability of observing the sequence $\vec{e}$ from that state given the input sequence $\vec{i}'$. The other parts can be explained analogously.

Proposition 11. *For all* PLTS, $s \in S$, $d \in D$, $q \in Q$, $r \in R$, $\vec{i}, \vec{i}' \in I^*$, *and* $\vec{e}, \vec{e}' \in E^*$,

$$\begin{aligned}
&\Pr[\, \lfloor [\![\langle L, s \rangle]\!](d{:}\vec{i}') \rfloor_E \sqsupseteq \vec{e}\,] \\
&\quad = \sum_{x \in S_\bot} \nu(x) \Pr[\, \lfloor [\![\langle L, x \rangle]\!](\vec{i}') \rfloor_E \sqsupseteq \vec{e}\,] && \text{where } s \stackrel{d}{\Rightarrow} \nu \\
&\Pr[\, \lfloor [\![\langle L, s \rangle]\!](q{:}\vec{i}') \rfloor_E \sqsupseteq q{:}\vec{e}\,'\,] \\
&\quad = \sum_{x \in S_\bot} \nu(x) \Pr[\, \lfloor [\![\langle L, x \rangle]\!](\vec{i}') \rfloor_E \sqsupseteq \vec{e}\,'\,] && \text{where } s \stackrel{q}{\Rightarrow} \nu \\
&\Pr[\, \lfloor [\![\langle L, s \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}\,'\,] \\
&\quad = \sum_{x \in S_\bot} \nu(x) \Pr[\, \lfloor [\![\langle L, x \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}\,'\,] && \text{where } s \stackrel{r}{\Rightarrow} \nu
\end{aligned}$$

Proof. For the first equality of the proposition: Note that if $\vec{e} = [\,]$, then

$$\Pr[\, \lfloor [\![\langle L, s \rangle]\!](d{:}\vec{i}') \rfloor_E \sqsupseteq \vec{e}\,] = 1 = \sum_{x \in S_\bot} \nu(x) \Pr[\, \lfloor [\![\langle L, x \rangle]\!](\vec{i}') \rfloor_E \sqsupseteq \vec{e}\,]$$

Otherwise, since $s \stackrel{d}{\Rightarrow} \nu$, we know there exists μ such that $s \stackrel{d}{\rightarrow} \mu$. It follows that

$$\begin{aligned}
\Pr[\, \lfloor [\![\langle L, s \rangle]\!](d{:}\vec{i}') \rfloor_E \sqsupseteq \vec{e}\,] &= \sum_{\vec{a} \in \gamma(\vec{e})} \Pr[\, [\![\langle L, s \rangle]\!](d{:}\vec{i}') \sqsupseteq \vec{a}\,] && (23) \\
&= \sum_{\vec{a}' \in \gamma(\vec{e})} \Pr[\, [\![\langle L, s \rangle]\!](d{:}\vec{i}') \sqsupseteq d{:}\vec{a}'\,] && (24) \\
&= \sum_{\vec{a}' \in \gamma(\vec{e})} \sum_{s' \in S} \langle L, s \rangle(d{:}\vec{i}')(d{:}\vec{a}', s') && (25) \\
&= \sum_{\vec{a}' \in \gamma(\vec{e})} \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') * \langle L, s'' \rangle(\vec{i}')(\vec{a}', s') && (26) \\
&= \sum_{x \in S_\bot} \nu(x) \Pr[\, \lfloor [\![\langle L, x \rangle]\!](\vec{i}') \rfloor_E \sqsupseteq \vec{e}\,] && (27)
\end{aligned}$$

Line 24 follows since $s \xrightarrow{d} \mu$ implies that s does not transition under any outputs and $\vec{e} \neq [\,]$ implies that $\vec{a} \in \gamma(\vec{e})$ cannot be $[\,]$. Thus, we know that the first action of $\vec{a}$ must be of the form $d{:}\vec{a}'$ for $\Pr[\,[\![\langle L, s\rangle]\!](d{:}\vec{i}') \sqsupseteq \vec{a}\,]$ to be non-zero. Since $\lfloor d{:}\vec{a}' \rfloor_E = \vec{e}$ and $d \notin E$, $\lfloor \vec{a}' \rfloor_E = \vec{e}$. Furthermore, $\mathsf{last}(\vec{a}') = \mathsf{last}(\vec{a}) = \mathsf{last}(\vec{e})$. Thus, $\vec{a}' \in \gamma(\vec{e})$. Line 27 follows from Proposition 10.

For the second equality of the proposition: Note that if $\vec{e}' = [\,]$, then

$$\Pr[\,\lfloor [\![\langle L, s\rangle]\!](q{:}\vec{i}') \rfloor_E \sqsupseteq q{:}\vec{e}'\,] = 1 = \sum_{x \in S_\perp} \nu(x) \Pr[\,\lfloor [\![\langle L, x\rangle]\!](\vec{i}') \rfloor_E \sqsupseteq \vec{e}'\,]$$

Otherwise, since $s \stackrel{q}{\Rightarrow} \nu$, we know there exists μ such that $s \xrightarrow{q} \mu$. It follows that

$$\begin{aligned}
\Pr[\,\lfloor [\![\langle L, s\rangle]\!](q{:}\vec{i}') \rfloor_E \sqsupseteq q{:}\vec{e}'\,] &= \sum_{\vec{a} \in \gamma(q:\vec{e}')} \Pr[\,[\![\langle L, s\rangle]\!](q{:}\vec{i}') \sqsupseteq \vec{a}\,] && (28)\\
&= \sum_{\vec{a}' \in \gamma(\vec{e}')} \Pr[\,[\![\langle L, s\rangle]\!](q{:}\vec{i}') \sqsupseteq q{:}\vec{a}'\,] && (29)\\
&= \sum_{\vec{a}' \in \gamma(\vec{e}')} \sum_{s' \in S} \langle L, s\rangle(q{:}\vec{i}')(q{:}\vec{a}', s') && (30)\\
&= \sum_{\vec{a}' \in \gamma(\vec{e}')} \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') * \langle L, s''\rangle(\vec{i}')(\vec{a}', s') && (31)\\
&= \sum_{x \in S_\perp} \nu(x) \Pr[\,\lfloor [\![\langle L, x\rangle]\!](\vec{i}') \rfloor_E \sqsupseteq \vec{e}'\,] && (32)
\end{aligned}$$

Line 29 follows since $s \xrightarrow{q} \mu$ implies that s does not transition under any outputs and $\vec{e}' \neq [\,]$ implies that $\vec{a}' \in \gamma(\vec{e}')$ cannot be $[\,]$. Thus, we know that the first action of $\vec{a}$ must be of the form $q{:}\vec{a}'$ for $\Pr[\,[\![\langle L, s\rangle]\!](q{:}\vec{i}') \sqsupseteq \vec{a}\,]$ to be non-zero. Since $\lfloor q{:}\vec{a}' \rfloor_E = q{:}\vec{e}'$, $\lfloor \vec{a}' \rfloor_E = \vec{e}'$. Furthermore, $\mathsf{last}(\vec{a}') = \mathsf{last}(\vec{a}) = \mathsf{last}(\vec{e}')$. Thus, $\vec{a}' \in \gamma(\vec{e}')$. Line 32 follows from Proposition 10.

For the third equality of the proposition: Note that if $\vec{e}' = [\,]$, then

$$\Pr[\,\lfloor [\![\langle L, s\rangle]\!](\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}'\,] = 1 = \sum_{x \in S_\perp} \nu(x) \Pr[\,\lfloor [\![\langle L, x\rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}'\,]$$

Otherwise, since $s \stackrel{r}{\Rightarrow} \nu$, we know there exists μ such that $s \xrightarrow{r} \mu$. It follows that

$$\begin{aligned}
\Pr[\,\lfloor [\![\langle L, s\rangle]\!](\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}'\,] &= \sum_{\vec{a} \in \gamma(r:\vec{e})} \Pr[\,[\![\langle L, s\rangle]\!](\vec{i}) \sqsupseteq \vec{a}\,] && (33)\\
&= \sum_{\vec{a}' \in \gamma(\vec{e}')} \Pr[\,[\![\langle L, s\rangle]\!](\vec{i}) \sqsupseteq r{:}\vec{a}'\,] && (34)\\
&= \sum_{\vec{a} \in \gamma(\vec{e}')} \sum_{s' \in S} \langle L, s\rangle(\vec{i})(r{:}\vec{a}', s') && (35)\\
&= \sum_{\vec{a} \in \gamma(\vec{e}')} \sum_{s' \in S} \sum_{s'' \in S} \mu(s'') * \langle L, s''\rangle(\vec{i})(\vec{a}', s') && (36)\\
&= \sum_{x \in S_\perp} \nu(x) \Pr[\,\lfloor [\![\langle L, x\rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}'\,] && (37)
\end{aligned}$$

Line 34 follows since $s \xrightarrow{r} \mu$ implies that s does not transition under any action other than r and $\vec{e}' \neq [\,]$ implies that $\vec{a}' \in \gamma(\vec{e}')$ cannot be $[\,]$. Thus, we know that the first action of $\vec{a}$ must be of the form $r{:}\vec{a}'$ for $\Pr[\,[\![\langle L, s\rangle]\!](\vec{i}) \sqsupseteq \vec{a}\,]$ to be non-zero. Since $\lfloor r{:}\vec{a}' \rfloor_E = r{:}\vec{e}'$, $\lfloor \vec{a}' \rfloor_E = \vec{e}'$. Furthermore, $\mathsf{last}(\vec{a}') = \mathsf{last}(\vec{a}) = \mathsf{last}(\vec{e}')$. Thus, $\vec{a}' \in \gamma(\vec{e}')$. Line 37 follows from Proposition 10. □

C Basic Properties of Differential Noninterference

Sequence Differencing. Given the input sequences $\vec{i}_1$ and $\vec{i}_2$, $\Delta(\vec{i}_1, \vec{i}_2)$ denotes the number of data points on which they differ: the minimum total number of data point insertions into $\vec{i}_1$ and $\vec{i}_2$ it takes to make them equal. Formally,

- $\Delta(\vec{i}_1, \vec{i}_2) = 0$ iff $\vec{i}_1 = \vec{i}_2$.
- For $1 \leq n$, $\Delta(\vec{i}_1, \vec{i}_2) = n$ iff there exists $d \in D$, $\vec{i}, \vec{i}'_1, \vec{i}'_2 \in I^*$, such that both of the following properties hold:
  - either $\vec{i}_1 = \vec{i}{:}d{:}\vec{i}'_1$ and $\vec{i}_2 = \vec{i}{:}\vec{i}'_2$, or $\vec{i}_1 = \vec{i}{:}\vec{i}'_1$ and $\vec{i}_2 = \vec{i}{:}d{:}\vec{i}'_2$; and
  - $\Delta(\vec{i}'_1, \vec{i}'_2) = n - 1$.

For $\Delta(\vec{i}_1, \vec{i}_2) = n$ to hold for any n, $\vec{i}_1$ and $\vec{i}_2$ must agree on every query from Q: they may only differ by n data points from D. Since differential privacy is defined using data sets differing on one element, in most theorems we are interested in the case where $\Delta(\vec{i}_1, \vec{i}_2) = 1$, which means that there exists $d \in D$, and $\vec{i}, \vec{i}' \in I^*$ such that either $\vec{i}_1 = \vec{i}{:}d{:}\vec{i}'$ and $\vec{i}_2 = \vec{i}{:}\vec{i}'$, or $\vec{i}_2 = \vec{i}{:}d{:}\vec{i}'$ and $\vec{i}_1 = \vec{i}{:}\vec{i}'$.

For example, let d_1 and d_2 range over elements in D, and q_1 and q_2 range over elements in Q.

- $\Delta([d_1, q_1, d_2], [d_1, q_1]) = 1$ (add d_2 to the end of the second sequence to get the first).
- $\Delta([q_1, d_2, q_2], [d_1, q_1, d_2, q_2]) = 1$ (add d_1 to the front of the first to get the second).
- $\Delta([d_1, q_1, d_2, q_2], [d_1, q_1, q_2]) = 1$ (add d_2 between q_1 and q_2 of the second to get the first).
- $\Delta([d_1, d_2, q_1, q_2], [d_1, d_2, q_2, q_1])$ is undefined (the two sequences do not agree on queries).

Note that in the first example, the two sequences have a difference of one under the above definition but do not have a Hamming distance since they are of different lengths.

While the choice of using all possible subsets of the set of trace prefixes instead of a single prefix makes the power of differential noninterference more apparent, it does not actually impose a stronger requirement as shown by the next lemma. This result simplifies reasoning about differential noninterference and is useful for proving subsequent results in this paper.

Proposition 12. *m has ϵ-differential noninterference if and only if for all input sequences $\vec{i}_1$ and $\vec{i}_2$ in I such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$ and $\vec{e}$ in E^*,*

$$\Pr[\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon) * \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}]$$

Proof. The only if direction follows directly from the definition by setting $S = \{\vec{e}\}$.

For the if direction, arbitrarily fix $\vec{i}_1$ and $\vec{i}_2$ such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$ and $S \subseteq E^*$. By assumption, for all $\vec{e}$ in E^*,

$$\Pr[\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon) * \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}]$$

Let S' be S with all the elements that are a longer version of another element of S removed. That is, $S' = \{\, \vec{e}' \in S \mid \nexists \vec{e} \in S \,\text{s.t.}\, \vec{e}' \sqsupset \vec{e} \,\}$ where $\vec{e}' \sqsupset \vec{e}$ means that $\vec{e}'$ is a strict prefix of $\vec{e}$. Proof by induction over the length of $\vec{e}$ shows that for all $\vec{e}$ in S, there exists $\vec{e}'$ in S' such that $\vec{e} \sqsupseteq \vec{e}'$. Thus, if there exists $\vec{e}$ in S such that $\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}$, then there exists $\vec{e}'$ in S' such that $\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}'$. Thus, for all $\vec{i}$, $\Pr[\lfloor m(\vec{i}) \rfloor_E \sqsupseteq S] = \Pr[\lfloor m(\vec{i}) \rfloor_E \sqsupseteq S']$.

For two $\vec{e}'_1$ and $\vec{e}'_2$ in S' such that $\vec{e}'_1 \neq \vec{e}'_2$, $\lfloor m(\vec{i}) \rfloor_E$ can only have one of them as a prefix since neither is a prefix of the other. Thus, since S is countable, this implies that

$$\Pr[\lfloor m(\vec{i}) \rfloor_E \sqsupseteq S] = \sum_{\vec{e}' \in S'} \Pr[\lfloor m(\vec{i}) \rfloor_E \sqsupseteq \vec{e}']$$

Thus,

$$\begin{aligned}
\Pr[\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq S] &= \Pr[\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq S'] \\
&= \sum_{\vec{e}' \in S'} \Pr[\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}'] \\
&\leq \sum_{\vec{e}' \in S'} \exp(\epsilon) * \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}'] \\
&= \exp(\epsilon) \sum_{\vec{e}' \in S'} \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}'] \\
&= \exp(\epsilon) \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq S'] \\
&= \exp(\epsilon) \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq S]
\end{aligned}$$

□

The next theorem is analogous to previous results about differential privacy for functions: it proves that the privacy leakage bound for a system whose inputs differ on at most n data points is $n * \epsilon$ where ϵ is the leakage bound for the system if its inputs differ on one data point (see e.g., corollary of [MT07]).

Proposition 13. *If a system m has ϵ-differential noninterference, then for all input sequences $\vec{i}_1$ and $\vec{i}_2$ such that $\Delta(\vec{i}_1, \vec{i}_2) \leq n$ and for all $S \subseteq E^*$,*

$$\Pr[\lfloor [\![M]\!](\vec{i}_1) \rfloor_E \sqsupseteq S] \leq \exp(n * \epsilon) * \Pr[\lfloor [\![M]\!](\vec{i}_2) \rfloor_E \sqsupseteq S]$$

Proof. Proof by induction over n.

Base Case: $n = 0$. In this case, $\vec{i}_1 = \vec{i}_2$ and, thus, $\Pr[\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq S] = \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq S]$ as needed with $\exp(0) = 1$.

Inductive Case: Assume for all $n' \leq n$; prove for $n + 1$. Since $\Delta(\vec{i}_1, \vec{i}_2) = n + 1$, there must exist $\vec{i}, \vec{i}'_1, \vec{i}'_2 \in I^*$ and $d_1 \in D$ such that $\vec{i}_1 = \vec{i}{:}d_1{:}\vec{i}'_1$, $\vec{i}_2 = \vec{i}{:}\vec{i}'_2$, and $\Delta(\vec{i}'_1, \vec{i}'_2) = n$. Let $\vec{i}_3 = \vec{i}{:}i_2{:}\vec{i}'_1$. $\Delta(\vec{i}_1, \vec{i}_3) = 1$ and $\Delta(\vec{i}_3, \vec{i}_2) = n$. Thus, by the inductive hypothesis,

$$\Pr[\lfloor m(\vec{i}_1) \rfloor_E \sqsupseteq S] \leq \exp(1 * \epsilon) * \Pr[\lfloor m(\vec{i}_3) \rfloor_E \sqsupseteq S]$$

and

$$\Pr[\lfloor m(\vec{i}_3) \rfloor_E \sqsupseteq S] \leq \exp(n * \epsilon) * \Pr[\lfloor m(\vec{i}_2) \rfloor_E \sqsupseteq S]$$

Thus,

$$\begin{aligned}\Pr[\lfloor m(\vec{i}_1)\rfloor_E \sqsupseteq S] &\leq \exp(1*\epsilon) * \Big(\exp(n*\epsilon) * \Pr[\lfloor m(\vec{i}_2)\rfloor_E \sqsupseteq S]\Big) \\ &= \exp(n+1*\epsilon) * \Pr[\lfloor m(\vec{i}_2)\rfloor_E \sqsupseteq S]\end{aligned}$$

as needed. □

D Compositional Reasoning

To prove Theorem 1, we use a definition and a proposition that helps us to track when the transition under $h^\dagger$ is being simulated by many transitions of M_2.

Let $L_1 = \langle S_1, Q_1 \uplus D_1, R_1 \uplus H_1, \rightarrow_1\rangle$ and $L_2 = \langle S_2, \emptyset, H_2, \rightarrow_2\rangle$ and $M_2 = \langle L_2, s_2^0\rangle$. Let $A_3 = Q_1 \uplus D_1 \uplus R_1 \uplus H_1 \uplus H_2 \uplus \{h^\ddagger\}$. Let $h^\dagger$ be a distinguished internal action in H_1. For simplicity, we assume that $h^\dagger$ only labels the one transition of L_1 that is implemented by M_2. Let $h^\ddagger$ be a distinguished internal action not in H_1 or H_2.

Let $\Psi(\vec{a})$ be a set of action sequences formed by replacing each action $h^\dagger$ in $\vec{a}$ with the internal action $h^\ddagger$ followed by any sequence $\vec{h}$ from H_2^+ and then $h^\ddagger$ again. Formally,

$$\begin{aligned}\Psi(h^\dagger{:}\vec{a}) &= h^\ddagger{:}H_2^+{:}h^\ddagger{:}\Psi(\vec{a}) \\ \Psi(a{:}\vec{a}) &= a{:}\Psi(\vec{a}) \qquad\qquad \text{where } a \neq h^\dagger\end{aligned}$$

where : is raised to work over sets in the standard way: for $X \subseteq A^*$ and $Y \subseteq A^*$, $X{:}Y = \{\, \vec{a} \in A^* \mid \exists \vec{a}_1 \in X, \exists \vec{a}_2 \in Y \text{ s.t. } \vec{a} = \vec{a}_1{:}\vec{a}_2 \,\}$ and $a{:}X = \{a\}{:}X$.

Proposition 14. *Let $M_1 = \langle L_1, s_0\rangle$ and let $M_3 = M_1[s^\dagger, M_2, \iota] = \langle L_3, s_0\rangle$ where M_2 implements the transition of $h^\dagger$ under ι. For all $\vec{a}$, $\vec{i}$, for all $s \in S$,*

$$\sum_{s' \in S_1} \langle L_1, s\rangle(\vec{i})(\vec{a}, s') = \sum_{\vec{a}' \in \Psi(a)} \sum_{s' \in S_1} \langle L_3, s\rangle(\vec{i})(\vec{a}', s')$$

Proof. We use induction over the structure of $\vec{a}$.

Case: $\vec{a} = [\,]$. Since $\Psi([\,]) = \{[\,]\}$, $\langle L_1, s\rangle(\vec{i})([\,], s') = \langle L_3, s\rangle(\vec{i})([\,], s') = 1$.

Case: $\vec{a} = i{:}\vec{a}'$.

- Subcase: there does not exist $\vec{i}'$ such that $\vec{i} = i{:}\vec{i}'$ and $s \xrightarrow{i}_1 \mu$. In this subcase, $\langle L_1, s\rangle(\vec{i})(\vec{a}, s') = 0$. By definition of Ψ we have that $\forall \vec{a}'' \in \Psi(\vec{a}) = \Psi(i{:}\vec{a}')$ and $\vec{a}''$ is of the form $i{:}\vec{a}'''$ for some $\vec{a}'''$. Since, by definition of $\rightarrow_3$, M_3 has an input transition from a state only if it has an input transition from that same state in M_1 there does not exist $s \xrightarrow{i}_3 \mu$. It follows that for all $\forall \vec{a}'' \in \Psi(\vec{a})$, $\langle L_3, s\rangle(\vec{i})(\vec{a}'', s') = 0$, as needed.

- Subcase: there does exist $\vec{i}'$ such that $\vec{i} = i{:}\vec{i}'$ and $s \xrightarrow{i}_1 \mu$. In this subcase,

$$\langle L_1, s\rangle(\vec{i})(\vec{a}, s') = \sum_{s'' \in S_1} \mu_1(s'')\langle L_1, s''\rangle(\vec{i}')(\vec{a}', s') \qquad \text{where } s \xrightarrow{i}_1 \mu_1$$

  and

$$\langle L_3, s\rangle(\vec{i})(\vec{a}, s') = \sum_{s'' \in S_1 \uplus S_2} \mu_2(s'')\langle L_3, s''\rangle(\vec{i}')(\vec{a}', s') \qquad \text{where } s \xrightarrow{i}_3 \mu_2.$$

Since each $\vec{a}'' \in \Psi(\vec{a})$ is of the form $i{:}\vec{a}'''$ for some $\vec{a}'''$, we need to show that

$$\sum_{s' \in S_1} \sum_{s'' \in S_1} \mu_1(s'') \langle L_1, s'' \rangle (\vec{i}')(\vec{a}', s') = \sum_{i:\vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \sum_{s'' \in S_1 \uplus S_2} \mu_2(s'') \langle L_3, s'' \rangle (\vec{i}')(\vec{a}''', s').$$

We reason as follows:

$$\begin{aligned}
&\sum_{i:\vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \sum_{s'' \in S_1 \uplus S_2} \mu_2(s'') \langle L_3, s'' \rangle (\vec{i}')(\vec{a}''', s') && (38)\\
&\qquad = \sum_{s'' \in S_1 \uplus S_2} \mu_2(s'') \sum_{i:\vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \langle L_3, s'' \rangle (\vec{i}')(\vec{a}''', s') && (39)\\
&\qquad = \sum_{s'' \in S_1} \mu_2(s'') \sum_{i:\vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \langle L_3, s'' \rangle (\vec{i}')(\vec{a}''', s') && (40)\\
&\qquad = \sum_{s'' \in S_1} \mu_2(s'') \sum_{\vec{a}''' \in \Psi(\vec{a}')} \sum_{s' \in S_1} \langle L_3, s'' \rangle (\vec{i}')(\vec{a}''', s') && (41)\\
&\qquad = \sum_{s'' \in S_1} \mu_1(s'') \sum_{\vec{a}''' \in \Psi(\vec{a}')} \sum_{s' \in S_1} \langle L_3, s'' \rangle (\vec{i}')(\vec{a}''', s') && (42)\\
&\qquad = \sum_{s' \in S_1} \sum_{s'' \in S_1} \mu_1(s'') \langle L_1, s'' \rangle (\vec{i}')(\vec{a}', s') && (43)
\end{aligned}$$

Line 39 follows from reordering summations and using distributivity of multiplication over summation. Line 40 follows from the fact that any $s'' \in \mathsf{Supp}(\mu_2)$ can not be in S_2 since any $s'' \in \mathsf{Supp}(\mu_2)$ is reachable via an input action. Line 41 follows from, the fact that each $\vec{a}'' \in \Psi(\vec{a})$ is of the form $i{:}\vec{a}'''$ and $\vec{a}''' \in \Psi(\vec{a}')$. Line 42 follows from definition of $\rightarrow_3$. We conclude in Line 43 using the inductive hypothesis $\sum_{s' \in S_1} \langle L_1, s'' \rangle (\vec{i}')(\vec{a}', s') = \sum_{\vec{a}''' \in \Psi(\vec{a}')} \sum_{s' \in S_1} \langle L_3, s'' \rangle (\vec{i}')(\vec{a}''', s')$.

Case: $\vec{a} = o{:}\vec{a}'$.

- Subcase: $o \neq h^\dagger$.
  - Subsubcase: there does not exist $\vec{a}'$ such that $\vec{a} = o{:}\vec{a}'$ and $s \xrightarrow{a}_1 \mu$. In this subcase, $\langle L_1, s \rangle (\vec{i})(\vec{a}, s') = 0$. By case definition we know $a = o{:}a'$ where $o \in R_1 \uplus H_1$ and $o \neq h^\dagger$. Then, $\forall \vec{a}'' \in \Psi(\vec{a}) = \Psi(o{:}\vec{a}')$, $\vec{a}''$ is of the form $o{:}\vec{a}'''$ for some $\vec{a}'''$. Since, by definition of $\rightarrow_3$, M_3 has an output transition on an action from $R_1 \uplus H_1 \setminus \{h^\dagger\}$ only if it has the same transition M_1. This gives $\langle L_3, s \rangle (\vec{i})(\vec{a}'', s') = 0$ for all $\forall \vec{a}'' \in \Psi(\vec{a})$, as needed.
  - Subsubcase: there does exist $\vec{a}'$ such that $\vec{a} = o{:}\vec{a}'$ and $s \xrightarrow{o}_1 \mu$. In this subcase, the proof follows a line of reasoning analogous to the case where $\vec{a} = i{:}\vec{a}'$. We show that

$$\sum_{s' \in S_1} \sum_{s'' \in S_1} \mu_1(s'') \langle L_1, s'' \rangle (\vec{i})(\vec{a}', s') = \sum_{o:\vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \sum_{s'' \in S_1 \uplus S_2} \mu_2(s'') \langle L_3, s'' \rangle (\vec{i})(\vec{a}''', s').$$

    In the step where we argue that $s'' \notin S_2$, we use the fact that all states in S_2 can only result from a transition on $h^\ddagger$ or an action from H_2, and that $o \in R_1 \uplus H_1 \setminus \{h^\dagger\}$, which is disjoint from $\{h^\ddagger\} \uplus H_2$.

- Subcase: $o = h^\dagger$.
  - Subsubcase: there does not exist μ such that $s \xrightarrow{h^\dagger}_1 \mu$, then $\langle L_1, s\rangle(\vec{i})(\vec{a}, s') = 0$ and $s \neq s^\dagger$. In this case all $\vec{a}'' \in \Psi(\vec{a})$ start with $h^\ddagger \vec{h} h^\ddagger$ for some vector $\vec{h}$ of actions from ${H_2}^+$. Since $s \neq s^\dagger$, according to the definition of $\rightarrow_3$ the only way for s to have a transition on $h^\ddagger$ is if $s = \iota(s_1)$ for some $s_1 \in \mathsf{Supp}(\mu^\dagger)$, where $s^\dagger \xrightarrow{h^\dagger}_1 \mu^\dagger$ and that $h^\ddagger$ transition is $\iota(s_1) \xrightarrow{h^\ddagger}_3 \mathsf{Dirac}(s_1)$. By definition of $\rightarrow_3$, s_1 can only transition on actions present in M_1, which means it cannot transition on any actions from H_2. This makes it impossible for $h^\ddagger$ to be followed by a sequence $\vec{h}$ from actions H_2, and gives $\langle L_3, s\rangle(\vec{i})(\vec{a}'', s') = 0$ for all $\vec{a}'' \in \Psi(\vec{a})$, as needed.
  - Subsubcase: there does exist μ such that $s \xrightarrow{h^\dagger}_1 \mu$, then by the assumption that $s^\dagger$ is the unique state enabling $h^\dagger$ we know that $s = s^\dagger$ and by transition-determinism $s^\dagger \xrightarrow{h^\dagger} \mu^\dagger$. We need to show that

$$\sum_{s' \in S_1} \sum_{s'' \in S_1} \mu^\dagger(s'') \langle L_1, s''\rangle(\vec{i})(\vec{a}', s') = \sum_{\vec{a}'' \in \Psi(\vec{a})} \sum_{s' \in S_1} \langle L_3, s^\dagger\rangle(\vec{i})(\vec{a}'', s').$$

    We reason as follows:

$$\sum_{\vec{a}'' \in \Psi(\vec{a})} \sum_{s' \in S_1} \langle L_3, s^\dagger\rangle(\vec{i})(\vec{a}'', s') \tag{44}$$

$$= \sum_{h^\ddagger : \vec{h} : h^\ddagger : \vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \langle L_3, s^\dagger\rangle(\vec{i})(h^\ddagger : \vec{h} : h^\ddagger : \vec{a}''', s') \tag{45}$$

$$= \sum_{h^\ddagger : \vec{h} : h^\ddagger \in a\vec{''} \in \Psi(\vec{a})} \sum_{s' \in S_1} \langle L_3, s_2^0\rangle(\vec{i})(\vec{h} : h^\ddagger : \vec{a}''', s') \tag{46}$$

$$= \sum_{h^\ddagger : \vec{h} : h^\ddagger \vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \sum_{s''' \in S_1 \uplus S_2} \langle L_3, s_2^0\rangle([\,])(\vec{h}, s''') * \langle L_3, s'''\rangle(\vec{i})(h^\ddagger : \vec{a}''', s') \tag{47}$$

$$= \sum_{h^\ddagger : \vec{h} : h^\ddagger \vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \sum_{s''' \in \iota(S^\dagger)} \langle L_3, s_2^0\rangle([\,])(\vec{h}, s''') * \langle L_3, s'''\rangle(\vec{i})(h^\ddagger : \vec{a}''', s') \tag{48}$$

$$= \sum_{h^\ddagger : \vec{h} : h^\ddagger \vec{a}''' \in \Psi(\vec{a})} \sum_{s' \in S_1} \sum_{s''' \in \iota(S^\dagger)} \langle L_3, s_2^0\rangle([\,])(\vec{h}, s''') * \langle L_3, s^4\rangle(\vec{i})(\vec{a}''', s') \tag{49}$$

$$\text{where } s''' \xrightarrow{h^\ddagger}_3 \mathsf{Dirac}(s^4) \tag{50}$$

$$= \sum_{s''' \in \iota(S^\dagger)} \Big(\sum_{\vec{h} \in (H_2)^+} \langle L_3, s^\dagger\rangle([\,])(\vec{h}, s''') \Big) * \sum_{s' \in S_1} \sum_{\vec{a}''' \in \Psi(\vec{a}')} \langle L_3, s^4\rangle(\vec{i})(\vec{a}''', s') \tag{51}$$

$$\text{where } s''' \xrightarrow{h^\ddagger}_3 \mathsf{Dirac}(s^4) \tag{52}$$

$$= \sum_{s''' \in \iota(S^\dagger)} \mu^\dagger(s''') \sum_{s' \in S_1} \sum_{\vec{a}''' \in \Psi(\vec{a}')} \langle L_3, s^4\rangle(\vec{i})(\vec{a}''', s') \tag{53}$$

    For Lines 44 to 47 we observe that by definition of Ψ each $\vec{a}'' \in \Psi(\vec{a})$ is of the form $h^\ddagger : \vec{h} : h^\ddagger : \vec{a}'''$ where $\vec{h} \neq [\,]$, by definition of $\rightarrow_3$, $s^\dagger \xrightarrow{h^\ddagger}_3 \mathsf{Dirac}(s_2^0)$ and use Proposition 6.

For Line 48 let $\iota(S^\dagger) = \{s \in L_2 \,|\, s = \iota(s') \text{ for some } s' \in \mathsf{Supp}(\mu^\dagger)\}$. The equation follows since $\langle L_3, s_2^0\rangle([\,])(\vec{h}, s''') = 0$ for all $s''' \in S_1$, by the assumption that for all $s \in \mathsf{Supp}(\mu^\dagger)$, $\mu^\dagger(s) = \sum_{\vec{h}\in(H_2)^+} M_2([\,])(\vec{h}, \iota(s))$ and that μ is a probability distribution.

By definition of $\rightarrow_3$ we know that $\iota(s_1) \xrightarrow{h^\ddagger}_3 \mathsf{Dirac}(s_1)$ for all $s_1 \in \mathsf{Supp}(\mu^\dagger)$ and Line 50 follows. Note that by definition of $\rightarrow_3$ and $\iota(S^\dagger)$ where since ι is an injection we know that for each $s''' \in \iota(S^\dagger)$ there is a unique state $s^4 \in \mathsf{Supp}(\mu^\dagger)$ such that $s''' = \iota(s^4)$. By using distributivity of multiplication over addition and the fact that $\vec{a'''} \in \Psi(\vec{a'})$ we get Line 52. Line 53 follows from the assumption that for all $s \in \mathsf{Supp}(\mu^\dagger)$,

$$\mu^\dagger(s) = \sum_{\vec{h}\in(H')^+} M'([\,])(\vec{h}, \iota(s)).$$

To conclude this case we recall that for each $s''' \in \iota(S^\dagger)$ there is a unique state $s^4 \in \mathsf{Supp}(\mu^\dagger)$ such that $s''' = \iota(s^4)$ and use the inductive hypothesis.

□

Proof of Theorem 1. Let $M_1 = \langle L_1, s_0\rangle$ and let $M_3 = M_1[s^\dagger, M_2, \iota] = \langle L_3, s_0\rangle$. We show that for all $\vec{i}$ in I^*, and $\vec{e}$ in E^*, s in S_1, $\Pr[\,\lfloor\langle L_1, s\rangle(\vec{i})\rfloor_E \sqsupseteq \vec{e}\,] = \Pr[\,\lfloor\langle L_3, s\rangle(\vec{i})\rfloor_E \sqsupseteq \vec{e}\,]$. By expanding the definitions of $\Pr[\,\lfloor\langle L_1, s\rangle(\vec{i})\rfloor_E \sqsupseteq \vec{e}\,]$ and $\Pr[\,\lfloor\langle L_3, s\rangle(\vec{i})\rfloor_E \sqsupseteq \vec{e}\,]$ we get

$$\begin{aligned}\Pr[\,\lfloor\langle L_1, s\rangle(\vec{i})\rfloor_E \sqsupseteq \vec{e}\,] &= \sum_{\vec{a}\in\gamma_1(\vec{e})} \Pr[\,\langle L_1, s\rangle(\vec{i}) \sqsupseteq \vec{a}\,] \\ &= \sum_{\vec{a}\in\gamma_1(\vec{e})} \sum_{s'\in S_1} \langle L_1, s\rangle(\vec{i})(\vec{a}, s')\end{aligned}$$

and

$$\begin{aligned}\Pr[\,\lfloor\langle L_3, s\rangle(\vec{i})\rfloor_E \sqsupseteq \vec{e}\,] &= \sum_{\vec{a}\in\gamma_3(\vec{e})} \Pr[\,\langle L_3, s\rangle(\vec{i}) \sqsupseteq \vec{a}\,] \\ &= \sum_{\vec{a}\in\gamma_3(\vec{e})} \sum_{s'\in S_1 \uplus S_2} \langle L_3, s\rangle(\vec{i})(\vec{a}, s')\end{aligned}$$

where γ_1 and γ_3 are sets of sequences of actions, respectively, of M_1 and M_3 defined as follows: $\gamma_1(\vec{e}) = \{\,\vec{a} \in A_1^* \mid \lfloor\vec{a}\rfloor_{E_1} = \vec{e} \wedge \mathsf{last}(\vec{a}) = \mathsf{last}(\vec{e})\,\}$ with the special case that $\gamma_1([\,]) = \{[\,]\}$, and $\gamma_3(\vec{e}) = \{\,\vec{a} \in A_3^* \mid \lfloor\vec{a}\rfloor_{E_3} = \vec{e} \wedge \mathsf{last}(\vec{a}) = \mathsf{last}(\vec{e})\,\}$ with the special case that $\gamma_3([\,]) = \{[\,]\}$ (as justified at the end of Appendix B). Note that we use A_i for the set of all actions of M_i and E_i for the set of observable actions of M_i.

Now we show that for all $s \in S_1$, $\vec{i}$, $\vec{e}$,

$$\sum_{\vec{a}\in\gamma_1(\vec{e})} \sum_{s'\in S_1} \langle L_1, s\rangle(\vec{i})(\vec{a}, s') = \sum_{\vec{a}\in\gamma_1(\vec{e})} \sum_{\vec{a}'\in\Psi(a)} \sum_{s'\in S_1} \langle L_3, s\rangle(\vec{i})(\vec{a'}, s') \tag{54}$$

$$= \sum_{\vec{a}\in\gamma_1(\vec{e})} \sum_{\vec{a}'\in\Psi(a)} \sum_{s'\in S_1\uplus S_2} \langle L_3, s\rangle(\vec{i})(\vec{a}', s') \tag{55}$$

$$= \sum_{\vec{a}\in\Phi(\vec{e})} \sum_{s'\in S_1\uplus S_2} \langle L_3, s\rangle(\vec{i})(\vec{a}, s') \tag{56}$$

$$= \sum_{\vec{a}\in\gamma_3(\vec{e})} \sum_{s'\in S_1\uplus S_2} \langle L_3, s\rangle(\vec{i})(\vec{a}, s') \tag{57}$$

where $\Phi(\vec{e}) = \bigcup_{\vec{a}\in\gamma_1(\vec{e})} \Psi(\vec{a})$, and Ψ is as defined at the top of this section.

Line 54 follows from Proposition 14.

For Line 55, we argue as follows: Since M_2 has no external actions and all transitions of M_3 on external actions end in a state in $S_1 \setminus S_2$, for those states $s' \in S_2$, s' is reachable via a hidden action only. Thus, for any $\vec{i}$, any $\vec{a} \in \gamma_3(\vec{e})$, $\langle L_3, s\rangle(\vec{i})(\vec{a}, s') = 0$ since $\vec{a}$ ends in an observable action from E by definition of γ_3.

Line 56 follows from the definition of Φ and the fact that for any pair of sequences $\vec{a_1}, \vec{a_2}$ such that $\vec{a_1} \neq \vec{a_2}$, $\Psi(\vec{a_1}) \cap \Psi(\vec{a_2}) = \emptyset$.

For Line 57 we observe that any sequence $\vec{a} \in \gamma_3(\vec{e}) \setminus \Phi(\vec{e})$ must have an occurrence of the action $h^\ddagger$ that is neither immediately preceded by a subsequence of the form $h^\ddagger{:}\vec{h}$ or immediately followed by a subsequence of the form $\vec{h}{:}h^\ddagger$. Then, by definition of $\rightarrow_3$, $\langle L_3, s\rangle(\vec{i})(\vec{a}, s') = 0$ for all sequences $\vec{a} \in \gamma_3(\vec{e}) \setminus \Phi(\vec{e})$, giving the needed equation.

E Proof of Soundness of Unwinding

E.1 A Helpful Proposition

Proposition 15. *If β is a bijection from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$ and for all $x_1' \in \mathsf{Supp}(\nu_1)$, $|\ln \nu_1(x_1') - \ln \nu_2(\beta(x_1'))| \leq \delta$, then*

$$\sum_{x_1'\in\mathsf{Supp}(\nu_1)} \nu_1(x_1') \exp(\epsilon' - \delta) \Pr[\, \lfloor [\![\langle L, \beta(x_1')\rangle]\!](\vec{i})\rfloor_E \sqsupseteq \vec{e}' \,]$$

$$\leq \exp(\epsilon') \sum_{x_2'\in S_\perp} \nu_2(x_2') \Pr[\, \lfloor [\![\langle L, x_2'\rangle]\!](\vec{i})\rfloor_E \sqsupseteq \vec{e}' \,]$$

Proof. For all x_1' in $\mathsf{Supp}(\nu_1)$,

$$\nu_1(x_1') \exp(\epsilon' - \delta) = \frac{\nu_1(x_1')}{\nu_2(\beta(x_1'))} \nu_2(\beta(x_1')) \exp(\epsilon' - \delta) \tag{58}$$

$$= \exp(\ln(\nu_1(x_1')) - \ln(\nu_2(\beta(x_1')))) \nu_2(\beta(x_1')) \exp(\epsilon' - \delta) \tag{59}$$

$$= \exp(\epsilon' - \delta + \ln(\nu_1(x_1')) - \ln(\nu_2(\beta(x_1')))) \nu_2(\beta(x_1')) \tag{60}$$

$$\leq \exp(\epsilon') \nu_2(\beta(x_1')) \tag{61}$$

Line 59 follows from the fact that for every $x_1' \in \mathsf{Supp}(\nu_1)$,

$$\nu_1(x_1')/\nu_2(\beta(x_1')) = \exp(\ln(\nu_1(x_1')))/\exp(\ln(\nu_2(\beta(x_1'))))$$

Line 61 follows since for every $x_1' \in \mathsf{Supp}(\nu_1)$, $|\ln \nu_1(x_1') - \ln \nu_2(\beta(x_1'))| \leq \delta$ implies that $\ln \nu_1(x_1') - \ln \nu_2(\beta(x_1')) \leq \delta$.

Thus,

$$\sum_{x_1' \in \mathsf{Supp}(\nu_1)} \nu_1(x_1') \exp(\epsilon' - \delta) \Pr[\, \lfloor [\![\langle L, \beta(x_1') \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] \tag{62}$$

$$\leq \sum_{x_1' \in \mathsf{Supp}(\nu_1)} \exp(\epsilon') \nu_2(\beta(x_1')) \Pr[\, \lfloor [\![\langle L, \beta(x_1') \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] \tag{63}$$

$$= \sum_{x_2' \in \mathsf{Supp}(\nu_2)} \exp(\epsilon') \nu_2(x_2') \Pr[\, \lfloor [\![\langle L, x_2' \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] \tag{64}$$

$$= \exp(\epsilon') \sum_{x_2' \in S_\perp} \nu_2(x_2') \Pr[\, \lfloor [\![\langle L, x_2' \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] \tag{65}$$

Line 64 follows from the fact that β is a bijection from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$. □

E.2 Proof of Lemma 1

Below we prove that $\Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}]$. Proving the reverse that $\Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}]$ is much the same reversing the roles of x_1 and x_2 and using β^{-1} in the place of β.

Proof by induction over the structures of $\vec{e}$ and $\vec{i}$.

Case: $\vec{e} = [\,]$. In this case,

$$\Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq [\,] \,] = 1 \leq \exp(\epsilon) * 1 = \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq [\,] \,]$$

Case: x_1 has no outgoing transitions and $\vec{e} \neq [\,]$. In this case, $\Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}] = 0 \leq \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq \vec{e}]$.

Henceforth, we only consider x_1 with at least one out going transition. Since x_1 is related to x_2, we know it must also have at least one out going transition. Thus, neither x_1 nor x_2 can be $\perp$. Thus, we use s_1 for x_1 and s_2 for x_2 for the reminder of the proof.

Case: $\vec{e} = q{:}\vec{e}'$ and $\vec{i} = [\,]$ for some $q \in Q$. In this case,

$$\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!](\vec{i}) \rfloor_E \sqsupseteq q{:}\vec{e}' \,] = \sum_{\vec{a} \in \gamma(q:\vec{e}')} \Pr[\, [\![\langle L, s_1 \rangle]\!](\vec{i}) \sqsupseteq \vec{a} \,] = \sum_{\vec{a} \in \gamma(q:\vec{e}')} \sum_{s_1' \in S} \langle L, s_1 \rangle(\vec{i})(\vec{a}, s_1')$$

Since $\vec{e} \neq [\,]$, $[\,]$ is not in $\gamma(\vec{e})$. Furthermore, all $\vec{a}$ in $\gamma(\vec{e})$ must have q come before any other action of E. In particular, $\vec{a}$ must have either the form $q{:}\vec{a}'$, $d{:}\vec{a}'$, or $h{:}\vec{a}'$ for some $\vec{a} \in A^*$, $d \in D$, and $h \in H$. Since s_1 is H-disabled by being in the unwinding relation, we know that for no $h \in H$ and μ does $s_1 \xrightarrow{h} \mu$. These factors combine to mean that $\langle L, s_1 \rangle([\,])(\vec{a}, s_1') = 0$ for all $s_1' \in S$ and $\vec{a} \in \gamma(\vec{e})$. Thus, $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!]([\,]) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = 0$. The same reasoning concludes that $\Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!]([\,]) \rfloor_E \sqsupseteq q{:}\vec{e}' \,] = 0$ making $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!]([\,]) \rfloor_E \sqsupseteq q{:}\vec{e}' \,] \leq \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!]([\,]) \rfloor_E \sqsupseteq q{:}\vec{e}' \,]$ since $0 \leq \exp(\epsilon) 0$

Case: $\vec{e} = r{:}\vec{e}'$ and $\vec{i} = [\,]$ for some $r \in R$. We consider the following subcases:

- Subcase: $s_1 \xrightarrow{r} \mu$ for some μ. Since s_1 and s_2 are related, there exists μ_2 such that $s_2 \xrightarrow{r} \mu_2$. This implies that there exists ν_1 and ν_2 such that $s_1 \xRightarrow{r} \nu_1$ and $s_2 \xRightarrow{r} \nu_2$. Since $s_1 \ \mathcal{R}^\epsilon \ s_2$, there exists δ in $[0, \epsilon]$ such that $\nu_1 \ \mathcal{L}(\mathcal{R}^{\epsilon-\delta}, \delta) \ \nu_2$. This implies there exists a bijection β from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$ such that for all $x_1 \in \mathsf{Supp}(\nu_1)$, $x_1 \mathcal{R}^{\epsilon-\delta} \beta(x_2)$ and $|\ln \nu_1(x_1) - \ln \nu_2(\beta(x_1))| \leq \delta$. Thus, we may apply the inductive hypothesis to $\vec{i}$ and $\vec{e}'$ to get for all x_1 in $\mathsf{Supp}(\nu_1)$, $\Pr[\, \lfloor [\![\langle L, x_1' \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] \leq \exp(\epsilon - \delta) \Pr[\, \lfloor [\![\langle L, \beta(x_1') \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,]$. Thus,

$$
\begin{aligned}
&\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] && (66)\\
&\quad = \sum_{x_1' \in S_\perp} \nu_1(x_1') \Pr[\, \lfloor [\![\langle L, x_1' \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] && (67)\\
&\quad = \sum_{x_1' \in \mathsf{Supp}(\nu_1)} \nu_1(x_1') \Pr[\, \lfloor [\![\langle L, x_1' \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] && (68)\\
&\quad \leq \sum_{x_1' \in \mathsf{Supp}(\nu_1)} \nu_1(x_1') \exp(\epsilon - \delta) \Pr[\, \lfloor [\![\langle L, \beta(x_1') \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] && (69)\\
&\quad \leq \exp(\epsilon) \sum_{x_2' \in S_\perp} \nu_2(x_2') \Pr[\, \lfloor [\![\langle L, x_2' \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq \vec{e}' \,] && (70)\\
&\quad = \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] && (71)
\end{aligned}
$$

  Lines 67 and 71 follow from Proposition 11. Line 69 follows from the inductive hypothesis Line 70 follows Proposition 15.

- Subcase: $s_1 \xrightarrow{r'} \mu$ for some output $r' \neq r$. Since s_1 and s_2 are related, there exists μ_2 such that $s_2 \xrightarrow{r'} \mu_2$. Furthermore, for no other action $a \neq r'$ does does $s_1 \xrightarrow{a} \mu'$ or $s_2 \xrightarrow{a} \mu'$ for any μ'. Recall that $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = \sum_{\vec{a} \in \gamma(r:\vec{e}')} \Pr[\, [\![\langle L, s_1 \rangle]\!] (\vec{i}) \sqsupseteq \vec{a} \,] = \sum_{\vec{a} \in \gamma(r:\vec{e}')} \sum_{s_1' \in S} \langle L, s_1 \rangle (\vec{i})(\vec{a}, s_1')$. For all $\vec{a} \in \gamma(r{:}\vec{e}')$, its first element from E must be r and, thus, it cannot start with r'. However, s can only transition under r' and $\vec{a} \neq [\,]$, meaning there must be a transition for $\vec{a}$ to be produced. Thus, for all such $\vec{a}$ and s_1', $\langle L, s_1 \rangle (\vec{i})(\vec{a}, s_1') = 0$ and $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = 0$. Similar reasoning concludes that $\Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = 0$. Thus, $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = 0 \leq \exp(\epsilon) * 0 = \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,]$ as needed.

- Subcase: s_1 is an input accepting state and $\vec{i} = [\,]$. Recall that

$$\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = \sum_{\vec{a} \in \gamma(r:\vec{e}')} \sum_{s_1' \in S} \langle L, s_1 \rangle (\vec{i})(\vec{a}, s_1')$$

  Since $\vec{a}$ cannot be $[\,]$, $\vec{i} = [\,]$, and s_1 is an input accepting state, this means that $\langle L, s_1 \rangle (\vec{i})(\vec{a}, s_1') = 0$ for all such $\vec{a}$ and s_1'. Thus, $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = 0$.

  Since s_1 is input accepting and related to s_2, s_2 must also be input accepting. Thus, by similar reasoning $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = 0$ and the results holds as above.

- Subcase: s_1 is an input accepting state and $\vec{i} = q{:}\vec{i}$ for some $q \in Q$. Since $\vec{e} = r{:}\vec{e}'$, no $\vec{a} \in \gamma(\vec{e})$ can have q come before r. Thus, much as above $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,] = 0 = \Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (\vec{i}) \rfloor_E \sqsupseteq r{:}\vec{e}' \,]$.

- Subcase: s_1 is an input accepting state and $\vec{i} = d{:}\vec{i'}$ for some $d \in D$. Since s_1 is input accepting and related to s_2, s_2 must also be input accepting. Thus, there exist ν_1 and ν_2 such that $s_1 \stackrel{d}{\Rightarrow} \nu_1$ and $s_2 \stackrel{d}{\Rightarrow} \nu_2$. Since $s_1 \ \mathcal{R}^\epsilon \ s_2$, there exists δ in $[0, \epsilon]$ such that $\nu_1 \ \mathcal{L}(\mathcal{R}^{\epsilon-\delta}, \delta) \ \nu_2$. This implies there exists a bijection β from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$ such that for all $x_1 \in \mathsf{Supp}(\nu_1)$, $x_1 \mathcal{R}^{\epsilon-\delta} \beta(x_2)$ and $|\ln \nu_1(x_1) - \ln \nu_2(\beta(x_1))| \leq \delta$. Thus, we may apply the inductive hypothesis to $\vec{i'}$ and $r{:}\vec{e'}$ to get for all x_1 in $\mathsf{Supp}(\nu_1)$, $\Pr[\,\lfloor [\![\langle L, x_1' \rangle]\!](\vec{i'})\rfloor_E \sqsupseteq r{:}\vec{e'}\,] \leq \exp(\epsilon - \delta) \Pr[\,\lfloor [\![\langle L, \beta(x_1') \rangle]\!](\vec{i'})\rfloor_E \sqsupseteq r{:}\vec{e'}\,]$. Thus,

$$\begin{aligned}
&\Pr[\,\lfloor [\![\langle L, s_1 \rangle]\!](d{:}\vec{i'})\rfloor_E \sqsupseteq r{:}\vec{e'}\,] && (72)\\
&\quad = \sum_{x_1' \in S_\perp} \nu_1(x_1') \Pr[\,\lfloor [\![\langle L, x_1' \rangle]\!](\vec{i'})\rfloor_E \sqsupseteq r{:}\vec{e'}\,] && (73)\\
&\quad = \sum_{x_1' \in \mathsf{Supp}(\nu_1)} \nu_1(x_1') \Pr[\,\lfloor [\![\langle L, x_1' \rangle]\!](\vec{i'})\rfloor_E \sqsupseteq r{:}\vec{e'}\,] && (74)\\
&\quad \leq \sum_{x_1' \in \mathsf{Supp}(\nu_1)} \nu_1(x_1') \exp(\epsilon - \delta) \Pr[\,\lfloor [\![\langle L, \beta(x_1') \rangle]\!](\vec{i'})\rfloor_E \sqsupseteq r{:}\vec{e'}\,] && (75)\\
&\quad \leq \exp(\epsilon) \sum_{x_2' \in S_\perp} \nu_2(x_2') \Pr[\,\lfloor [\![\langle L, x_2' \rangle]\!](\vec{i'})\rfloor_E \sqsupseteq r{:}\vec{e'}\,] && (76)\\
&\quad = \exp(\epsilon) \Pr[\,\lfloor [\![\langle L, s_2 \rangle]\!](d{:}\vec{i})\rfloor_E \sqsupseteq r{:}\vec{e'}\,] && (77)
\end{aligned}$$

  Lines 73 and 77 follow from Proposition 11. Line 75 follows from the inductive hypothesis Line 76 follows Proposition 15.

Case: $\vec{e} = q{:}\vec{e'}$ and $\vec{i} = i{:}\vec{i'}$ for some i in I and $\vec{i'}$ in I^*. We consider the following subcases:

- Subcase: s_1 is not an input accepting state: there exists no μ_1 such that $s_1 \stackrel{q}{\to} \mu_1$. Since s_1 and s_2 are related, there also cannot exist a μ_2 such that $s_2 \stackrel{q}{\to} \mu_2$. Since s_1 does have a transition and is H-disabled, there must exist some response r such that $s_1 \stackrel{r}{\to} \mu_1'$ and $s_2 \stackrel{r}{\to} \mu_2'$ for some μ_1' and μ_2'. Furthermore, s_1 and s_2 transitions under no other actions. Recall that

$$\begin{aligned}
\Pr[\,\lfloor [\![\langle L, s_1 \rangle]\!](i{:}\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] &= \sum_{\vec{a} \in \gamma(q:\vec{e'})} \Pr[\,[\![\langle L, s_1 \rangle]\!](i{:}\vec{i'}) \sqsupseteq \vec{a}\,]\\
&= \sum_{\vec{a} \in \gamma(q:\vec{e'})} \sum_{s_1' \in S} \langle L, s_1 \rangle(i{:}\vec{i'})(\vec{a}, s_1')
\end{aligned}$$

  For all $\vec{a} \in \gamma(q{:}\vec{e'})$, its first element from E must be q and, thus, it cannot start with r. However, s can only transition under r and $\vec{a} \neq [\,]$, meaning there must be a transition for $\vec{a}$ to be produced. Thus, for all such $\vec{a}$ and s_1', $\langle L, s_1 \rangle(\vec{i})(\vec{a}, s_1') = 0$ and $\Pr[\,\lfloor [\![\langle L, s_1 \rangle]\!](i{:}\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] = 0$. Similar reasoning concludes that $\Pr[\,\lfloor [\![\langle L, s_2 \rangle]\!](i{:}\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] = 0$. Thus,

$$\Pr[\,\lfloor [\![\langle L, s_1 \rangle]\!](i{:}\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] \leq \exp(\epsilon) * 0 = \exp(\epsilon) \Pr[\,\lfloor [\![\langle L, s_2 \rangle]\!](q{:}\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,]$$

  as needed.

- Subcase: s_1 is an input accepting state and $i = q$ for some μ_1. Since s_1 is input accepting, $s_1 \stackrel{q}{\to} \mu_1$ for some μ_1. Since s_1 and s_2 are related, there exists μ_2 such that $s_2 \stackrel{q}{\to} \mu_2$. This

implies that there exists ν_1 and ν_2 such that $s_1 \stackrel{q}{\Rightarrow} \nu_1$ and $s_2 \stackrel{q}{\Rightarrow} \nu_2$. Since $s_1 \; \mathcal{R}^\epsilon \; s_2$, there exists δ in $[0, \epsilon]$ such that $\nu_1 \; \mathcal{L}(\mathcal{R}^{\epsilon-\delta}, \delta) \; \nu_2$. This implies there exists a bijection β from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$ such that for all $x_1 \in \mathsf{Supp}(\nu_1)$, $x_1 \mathcal{R}^{\epsilon-\delta} \beta(x_2)$ and $|\ln \nu_1(x_1) - \ln \nu_2(\beta(x_1))| \leq \delta$. Thus, we may apply the inductive hypothesis to $\vec{i'}$ and $\vec{e'}$ to get for all x_1 in $\mathsf{Supp}(\nu_1)$, $\Pr[\, \lfloor [\![\langle L, x_1' \rangle]\!] (\vec{i'}) \rfloor_E \sqsupseteq \vec{e'} \,] \leq \exp(\epsilon - \delta) \Pr[\, \lfloor [\![\langle L, \beta(x_1') \rangle]\!] (\vec{i'}) \rfloor_E \sqsupseteq \vec{e'} \,]$. Thus,

$$\begin{aligned}
&\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (q{:}\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,] && (78)\\
&\quad = \sum_{x_1' \in S_\perp} \nu_1(x_1') \Pr[\, \lfloor [\![\langle L, x_1' \rangle]\!] (\vec{i'}) \rfloor_E \sqsupseteq \vec{e'} \,] && (79)\\
&\quad = \sum_{x_1' \in \mathsf{Supp}(\nu_1)} \nu_1(x_1') \Pr[\, \lfloor [\![\langle L, x_1' \rangle]\!] (\vec{i'}) \rfloor_E \sqsupseteq \vec{e'} \,] && (80)\\
&\quad \leq \sum_{x_1' \in \mathsf{Supp}(\nu_1)} \nu_1(x_1') \exp(\epsilon - \delta) \Pr[\, \lfloor [\![\langle L, \beta(x_1') \rangle]\!] (\vec{i'}) \rfloor_E \sqsupseteq \vec{e'} \,] && (81)\\
&\quad \leq \exp(\epsilon) \sum_{x_2' \in S_\perp} \nu_2(x_2') \Pr[\, \lfloor [\![\langle L, x_2' \rangle]\!] (\vec{i'}) \rfloor_E \sqsupseteq \vec{e'} \,] && (82)\\
&\quad = \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!] (q{:}\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,] && (83)
\end{aligned}$$

Lines 79 and 83 follow from Proposition 11. Line 81 follows from the inductive hypothesis Line 82 follows Proposition 15.

- Subcase: s_1 is input accepting, $i \neq q$, and $i \in Q$. Recall that

$$\begin{aligned}
\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (i{:}\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,] &= \sum_{\vec{a} \in \gamma(q:\vec{e'})} \Pr[\, [\![\langle L, s_1 \rangle]\!] (i{:}\vec{i'}) \sqsupseteq \vec{a} \,] \\
&= \sum_{\vec{a} \in \gamma(q:\vec{e'})} \sum_{s_1' \in S} \langle L, s_1 \rangle (i{:}\vec{i'}) (\vec{a}, s_1')
\end{aligned}$$

  For all $\vec{a} \in \gamma(q{:}\vec{e'})$, its first element from E must be q and, thus, it cannot start with i. Thus, for all such $\vec{a}$ and s_1', $\langle L, s_1 \rangle (i{:}\vec{i'})(\vec{a}, s_1') = 0$ and $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (i{:}\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,] = 0$. Similar reasoning allows us to conclude that $\Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!] (i{:}\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,] = 0$. Thus, $\Pr[\, \lfloor [\![\langle L, s_1 \rangle]\!] (i{:}\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,] = 0 \leq \exp(\epsilon) * 0 = \exp(\epsilon) \Pr[\, \lfloor [\![\langle L, s_2 \rangle]\!] (q{:}\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,]$ as needed.

- Subcase: s_1 is input accepting, $i \neq q$, and $i \in D$. We use d to denote i. Since s_1 is input accepting and related to s_2, s_2 must also be input accepting. Thus, there exist ν_1 and ν_2 such that $s_1 \stackrel{d}{\Rightarrow} \nu_1$ and $s_2 \stackrel{d}{\Rightarrow} \nu_2$. Since $s_1 \; \mathcal{R}^\epsilon \; s_2$, there exists δ in $[0, \epsilon]$ such that $\nu_1 \; \mathcal{L}(\mathcal{R}^{\epsilon-\delta}, \delta) \; \nu_2$. This implies there exists a bijection β from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$ such that for all $x_1 \in \mathsf{Supp}(\nu_1)$, $x_1 \mathcal{R}^{\epsilon-\delta} \beta(x_2)$ and $|\ln \nu_1(x_1) - \ln \nu_2(\beta(x_1))| \leq \delta$. Thus, we may apply the inductive hypothesis to $\vec{i'}$ and $q{:}\vec{e'}$ to get for all x_1 in $\mathsf{Supp}(\nu_1)$, $\Pr[\, \lfloor [\![\langle L, x_1' \rangle]\!] (\vec{i'}) \rfloor_E \sqsupseteq q{:}\vec{e'} \,] \leq$

$\exp(\epsilon-\delta)\Pr[\,\lfloor[\![\langle L,\beta(x_1')\rangle]\!](\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,]$. Thus,

$$\begin{aligned}
&\Pr[\,\lfloor[\![\langle L,s_1\rangle]\!](d{:}\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] && (84)\\
&\quad= \sum_{x_1'\in S_\perp} \nu_1(x_1')\Pr[\,\lfloor[\![\langle L,x_1'\rangle]\!](\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] && (85)\\
&\quad= \sum_{x_1'\in \mathsf{Supp}(\nu_1)} \nu_1(x_1')\Pr[\,\lfloor[\![\langle L,x_1'\rangle]\!](\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] && (86)\\
&\quad\le \sum_{x_1'\in \mathsf{Supp}(\nu_1)} \nu_1(x_1')\exp(\epsilon-\delta)\Pr[\,\lfloor[\![\langle L,\beta(x_1')\rangle]\!](\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] && (87)\\
&\quad\le \exp(\epsilon)\sum_{x_2'\in S_\perp} \nu_2(x_2')\Pr[\,\lfloor[\![\langle L,x_2'\rangle]\!](\vec{i'})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] && (88)\\
&\quad= \exp(\epsilon)\Pr[\,\lfloor[\![\langle L,s_2\rangle]\!](d{:}\vec{i})\rfloor_E \sqsupseteq q{:}\vec{e'}\,] && (89)
\end{aligned}$$

Lines 85 and 89 follow from Proposition 11. Line 87 follows from the inductive hypothesis Line 88 follows Proposition 15.

E.3 Proof of Theorem 2

We use Lemma 12 and strengthen the hypothesis to show that for all reachable states s and $\vec{e}$,

$$\begin{aligned}
\Pr[\,\lfloor[\![\langle L,s\rangle]\!](\vec{i_1})\rfloor_E \sqsupseteq \vec{e}\,] &= \sum_{\vec{a}\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s\rangle]\!](\vec{i_1})\sqsupseteq\vec{a}\,]\\
&\le \exp(\epsilon)\sum_{\vec{a}\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s\rangle]\!](\vec{i_2})\sqsupseteq\vec{a}\,]\\
&= \exp(\epsilon)\Pr[\,\lfloor[\![\langle L,s\rangle]\!](\vec{i_2})\rfloor_E \sqsupseteq \vec{e}\,]
\end{aligned}$$

Arbitrarily fix $\vec{i_1}$ and $\vec{i_2}$ such that $\Delta(\vec{i_1},\vec{i_2})=1$. We use induction over the structures of $\vec{i_1}$, $\vec{i_2}$, and $\vec{e}$.

Case: $\vec{e}=[\,]$. In this case, $\gamma(\vec{e})=\{[\,]\}$ and $\Pr[\,\lfloor[\![\langle L,s\rangle]\!](\vec{i_1})\rfloor_E \sqsupseteq [\,]\,] = 1 \le \exp(\epsilon)*1 = \exp(\epsilon)\Pr[\,\lfloor[\![\langle L,s\rangle]\!](\vec{i_2})\rfloor_E \sqsupseteq [\,]\,]$ irrespective of $\vec{i_1}$ and $\vec{i_2}$.

Only in the case where $\vec{e}=[\,]$, can $[\,]$ be in $\gamma(\vec{e})$. Thus, we assume that $\vec{e}\neq[\,]$ in the reminder of this proof.

Case: $\vec{i_1}=[\,]$ and $\vec{i_2}=[\,]$. $\Pr[\,[\![\langle L,s\rangle]\!]([\,])\sqsupseteq\vec{a}\,] = \Pr[\,[\![\langle L,s\rangle]\!]([\,])\sqsupseteq\vec{a}\,]$ for all $\vec{a}\in\gamma(\vec{e})$ for any $\vec{e}$.

Case: $\vec{i_1}=d{:}\vec{i_1'}$ and $\vec{i_2}=d{:}\vec{i_2'}$. We consider three mutually exclusive subcases:

- Subcase: $s\xrightarrow{d}\mu$. For all $\vec{a}$ such that for no $\vec{a'}$, $\vec{a}=d{:}\vec{a'}$, $\Pr[\,[\![\langle L,s\rangle]\!](\vec{i_1})\sqsupseteq\vec{a}\,]=0=\Pr[\,[\![\langle L,s\rangle]\!](\vec{i_2})\sqsupseteq\vec{a}\,]$. Since such $\vec{a}$ add nothing to the summations, we may ignore them and limit our attention to $\vec{a}=d{:}\vec{a'}$ in $\gamma(\vec{e})$. Note that all such $\vec{a'}$ are in $\gamma(\vec{e})$ iff $d{:}\vec{a'}$ is in $\gamma(\vec{e})$.

  All the states in $\mathsf{Supp}(\mu)$ are reachable. Thus, for each state s' in $\mathsf{Supp}(\mu)$, we may apply the inductive hypothesis on $\vec{i_1'}$, $\vec{i_2'}$, and $\vec{e}$ to get that

$$\sum_{\vec{a'}\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s'\rangle]\!](\vec{i_1'})\sqsupseteq\vec{a'}\,] \le \exp(\epsilon)\sum_{\vec{a'}\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s'\rangle]\!](\vec{i_2'})\sqsupseteq\vec{a'}\,]$$

Considering the sum over all such $\vec{a}$, we get

$$\sum_{\vec{a}\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s\rangle]\!](\vec{i}_1)\sqsupseteq\vec{a}\,] \tag{90}$$

$$= \sum_{d:\vec{a}'\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s\rangle]\!](\vec{i}_1)\sqsupseteq d{:}\vec{a}'\,] \tag{91}$$

$$= \sum_{\vec{a}'\in\gamma(\vec{e})} \sum_{s'\in S} \mu(s')\Pr[\,[\![\langle L,s'\rangle]\!](\vec{i}_1')\sqsupseteq\vec{a}'\,] \tag{92}$$

$$= \sum_{s'\in S} \mu(s') \sum_{\vec{a}'\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s'\rangle]\!](\vec{i}_1')\sqsupseteq\vec{a}'\,] \tag{93}$$

$$= \sum_{s'\in \mathsf{Supp}(\mu)} \mu(s') \sum_{\vec{a}'\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s'\rangle]\!](\vec{i}_1')\sqsupseteq\vec{a}'\,] \tag{94}$$

$$\leq \sum_{s'\in \mathsf{Supp}(\mu)} \mu(s')\exp(\epsilon) \sum_{\vec{a}'\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s'\rangle]\!](\vec{i}_2')\sqsupseteq\vec{a}'\,] \tag{95}$$

$$= \exp(\epsilon) \sum_{\vec{a}'\in\gamma(\vec{e})} \sum_{s'\in S} \mu(s')\Pr[\,[\![\langle L,s'\rangle]\!](\vec{i}_2')\sqsupseteq\vec{a}'\,] \tag{96}$$

$$= \exp(\epsilon) \sum_{d:\vec{a}'\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s\rangle]\!](\vec{i}_2)\sqsupseteq d{:}\vec{a}'\,] \tag{97}$$

$$= \exp(\epsilon) \sum_{\vec{a}\in\gamma(\vec{e})} \Pr[\,[\![\langle L,s\rangle]\!](\vec{i}_2)\sqsupseteq\vec{a}\,] \tag{98}$$

where $d{:}\vec{a}'$ in the expression $d{:}\vec{a}' \in \gamma(\vec{e})$ ranges over only those elements of $\gamma(\vec{e})$ of the form $d{:}\vec{a}'$. That is, $\sum_{d:\vec{a}'\in\gamma(\vec{e})}$ is shorthand for

$$\sum_{d:\vec{a}'\in\{\,\vec{a}''\in\gamma(\vec{e})\;\mid\;\exists\vec{a}'\in A^*\text{ s.t. }d:\vec{a}'=\vec{a}''\,\}}$$

Note that the last line follows from the fact that $[\![\langle L,s\rangle]\!](\vec{i}_2)(\vec{a}) = 0$ for all $\vec{a}$ not of the form $d{:}\vec{a}'$. Lines 92 and 97 follow from Proposition 9. Line 95 follows from the inductive hypothesis.

- Subcase: $s \xrightarrow{r} \mu$ for some $r \in R$. For all $\vec{a}$ such that for no $\vec{a}'$, $\vec{a} = r{:}\vec{a}'$, $\Pr[\,[\![\langle L,s\rangle]\!](\vec{i}_1)\sqsupseteq\vec{a}\,] = 0 = \Pr[\,[\![\langle L,s\rangle]\!](\vec{i}_2)\sqsupseteq\vec{a}\,]$. Since such $\vec{a}$ add nothing to the summations, we may ignore them and limit our attention to $r{:}\vec{a}'$ in $\gamma(\vec{e})$. Unless $\vec{e} = r{:}\vec{e}'$ for some $\vec{e}'$, no such $r{:}\vec{a}'$ will be in $\gamma(\vec{e})$ and both summations will be zero. Thus, we limit our attention to the case where $\vec{e} = r{:}\vec{e}'$ for some $\vec{e}'$. In this case, we may use the inductive hypothesis on $\vec{i}_1$, $\vec{i}_2$, and $\vec{e}'$ to get that for

all $s' \in \mathsf{Supp}(\mu)$, $\sum_{\vec{a}' \in \gamma(\vec{e}')} \Pr[\, [\![\langle L, s' \rangle]\!](\vec{i}_1) \sqsupseteq \vec{a}' \,] \leq \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e}')} \Pr[\, [\![\langle L, s' \rangle]\!](\vec{i}_2) \sqsupseteq \vec{a}' \,]$. Thus,

$$\sum_{\vec{a} \in \gamma(\vec{e})} \Pr[\, [\![\langle L, s \rangle]\!](\vec{i}_1) \sqsupseteq \vec{a} \,] \tag{99}$$

$$= \sum_{r:\vec{a}' \in \gamma(r:\vec{e}')} \Pr[\, [\![\langle L, s \rangle]\!](\vec{i}_1) \sqsupseteq r{:}\vec{a}' \,] \tag{100}$$

$$= \sum_{\vec{a}' \in \gamma(\vec{e}')} \sum_{s' \in S} \mu(s') \Pr[\, [\![\langle L, s' \rangle]\!](\vec{i}_1) \sqsupseteq \vec{a}' \,] \tag{101}$$

$$= \sum_{s' \in \mathsf{Supp}(\mu)} \mu(s') \sum_{\vec{a}' \in \gamma(\vec{e}')} \Pr[\, [\![\langle L, s' \rangle]\!](\vec{i}_1) \sqsupseteq \vec{a}' \,] \tag{102}$$

$$\leq \sum_{s' \in \mathsf{Supp}(\mu)} \mu(s') \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e}')} \Pr[\, [\![\langle L, s' \rangle]\!](\vec{i}_2) \sqsupseteq \vec{a}' \,] \tag{103}$$

$$= \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e}')} \sum_{s' \in S} \mu(s') \Pr[\, [\![\langle L, s' \rangle]\!](\vec{i}_2) \sqsupseteq \vec{a}' \,] \tag{104}$$

$$= \exp(\epsilon) \sum_{r:\vec{a}' \in \gamma(r:\vec{e}')} \Pr[\, [\![\langle L, s \rangle]\!](\vec{i}_2) \sqsupseteq r{:}\vec{a}' \,] \tag{105}$$

$$= \exp(\epsilon) \sum_{\vec{a} \in \gamma(\vec{e})} \Pr[\, [\![\langle L, s \rangle]\!](\vec{i}_2) \sqsupseteq \vec{a} \,] \tag{106}$$

Lines 101 and 105 follow from Proposition 9. Line 103 follows from the inductive hypothesis.

- Subcase: Otherwise. Since s is H-disabled, it is not the case that $s \xrightarrow{h} \mu$ for any μ or $h \in H$. Since $\vec{a} \neq [\,]$, $\Pr[\, [\![\langle L, s \rangle]\!](\vec{i}_1) \sqsupseteq \vec{a} \,] = 0 = \Pr[\, [\![\langle L, s \rangle]\!](\vec{i}_2) \sqsupseteq \vec{a} \,]$ for all $\vec{a} \in \gamma(\vec{e})$.

Case: $\vec{i}_1 = q{:}\vec{i}_1'$ and $\vec{i}_2 = q{:}\vec{i}_2'$. Much as above just using that $\vec{a}$ is only in $\gamma(\vec{e})$ if $\vec{e} = q{:}\vec{e}'$ for some $\vec{e}'$ and $\vec{a}' \in \gamma(\vec{e}')$.

Case: $\vec{i}_2 = d{:}\vec{i}_1$. We consider the following subcases:

- Subcase: $s \xrightarrow{d} \mu$. Since $s \xrightarrow{d} \mu$, for some ν, $s \stackrel{d}{\Rightarrow} \nu$. Since s is reachable from s_0, there exists an ϵ-unwinding relation $\mathcal{R}^\epsilon$ that covers s and d. That is, for all $s' \in \mathsf{Supp}(\nu)$, for all $s' \in \mathsf{Supp}(\nu)$,

$s \mathcal{R}^\epsilon s'$ and $\nu(\bot) = 0$.

$$\Pr[\lfloor [\![\langle L, s\rangle]\!] \rfloor_E(\vec{i}_1) \sqsupseteq \vec{e}] \leq \exp(\epsilon) \Pr[\lfloor [\![\langle L, s_{\mathsf{min}}\rangle]\!] \rfloor_E(\vec{i}_1) \sqsupseteq \vec{e}] \tag{107}$$

$$= \exp(\epsilon) \left(\sum_{x \in S_\bot} \nu(x) \right) \Pr[\lfloor [\![\langle L, s_{\mathsf{min}}\rangle]\!](\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] \tag{108}$$

$$= \exp(\epsilon) \left(\sum_{s' \in S} \nu(s') \right) \Pr[\lfloor [\![\langle L, s_{\mathsf{min}}\rangle]\!](\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] \tag{109}$$

$$\leq \exp(\epsilon) \sum_{s' \in S} \nu(s') \Pr[\lfloor [\![\langle L, s'\rangle]\!](\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] \tag{110}$$

$$= \exp(\epsilon) \sum_{x \in S_\bot} \nu(x) \Pr[\lfloor [\![\langle L, x\rangle]\!](\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] \tag{111}$$

$$= \exp(\epsilon) \Pr[\lfloor [\![\langle L, s\rangle]\!](d{:}\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] \tag{112}$$

$$= \exp(\epsilon) \Pr[\lfloor [\![\langle L, s\rangle]\!](\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}] \tag{113}$$

where s_{min} is the state $s' \in \mathsf{Supp}(\nu)$ that minimizes $\Pr[\lfloor [\![\langle L, s'\rangle]\!](\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}]$. Line 107 follows from Lemma 1. Line 108 follows from Proposition 7. Lines 109 and 111 follow from $\nu(\bot) = 0$. Line 112 follows from Proposition 11.

- Subcase: $s \xrightarrow{r} \mu$ for some r. As in the corresponding subcase in the case for $\vec{i}_1 = d{:}\vec{i}'_1$ and $\vec{i}_2 = d{:}\vec{i}'_2$, we may ignore $\vec{a}$ not of the form $\vec{a} = r{:}\vec{a}'$ and $\vec{e}$ not of the form $r{:}\vec{e}'$. In this case, we may use the inductive hypothesis on $\vec{i}_1$, $\vec{i}_2$, and $\vec{e}'$ as before to get the required result.

- Subcase: Otherwise. Since s does not transition under d in this case and the automaton has quasi-input enabling, it does not transition under any input action. Further, s is H-disabled. Thus, since $\vec{a} \neq [\,]$, $\Pr[[\![\langle L, s\rangle]\!](\vec{i}_1) \sqsupseteq \vec{a}\,] = 0$ for all $\vec{a} \in \gamma(\vec{e})$.

Case: $\vec{i}_1 = d{:}\vec{i}_2$. We consider the following subcases.

- Subcase: $s \xrightarrow{d} \mu$. Since $s \xrightarrow{d} \mu$, for some ν, $s \stackrel{d}{\Rightarrow} \nu$. Since s is reachable from s_0, there exists an ϵ-unwinding relation $\mathcal{R}^\epsilon$ that covers s and d. That is, for all $s' \in \mathsf{Supp}(\nu)$, for all $s' \in \mathsf{Supp}(\nu)$, $s \mathcal{R}^\epsilon s'$ and $\nu(\bot) = 0$.

  Thus,

$$\Pr[\lfloor [\![\langle L, s\rangle]\!](\vec{i}_1) \rfloor_E \sqsupseteq \vec{e}] = \Pr[\lfloor [\![\langle L, s\rangle]\!](d{:}\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}] \tag{114}$$

$$= \sum_{x \in S_\bot} \nu(x) \Pr[\lfloor [\![\langle L, x\rangle]\!](\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}] \tag{115}$$

$$= \sum_{x \in \mathsf{Supp}(\nu)} \nu(x) \Pr[\lfloor [\![\langle L, x\rangle]\!](\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}] \tag{116}$$

$$\leq \sum_{x \in \mathsf{Supp}(\nu)} \nu(x) \exp(\epsilon) \Pr[\lfloor [\![\langle L, s\rangle]\!](\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}] \tag{117}$$

$$= \left(\sum_{x \in \mathsf{Supp}(\nu)} \nu(x) \right) \exp(\epsilon) \Pr[\lfloor [\![\langle L, s\rangle]\!](\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}] \tag{118}$$

$$= \exp(\epsilon) \Pr[\lfloor [\![\langle L, s\rangle]\!](\vec{i}_2) \rfloor_E \sqsupseteq \vec{e}] \tag{119}$$

Line 115 follows from Proposition 11. Line 117 follows from Lemma 1. Line 119 follows from Proposition 7.

- Subcase: $s \xrightarrow{r} \mu$ for some r. As above in the other subcases for $s \xrightarrow{r} \mu$.

- Subcase: Otherwise. In the case where $s \xrightarrow{d} \mu$ for no μ and $\vec{a} \neq [\,]$, everything is 0, which is lower than any possible value of $\exp(\epsilon)\Pr[\,\lfloor [\![\langle L, s\rangle]\!](\vec{i}_2)\rfloor_E \sqsupseteq \vec{e}]$.

F Proof of Lemma 2: $M_{\mathsf{ex1}}(K)$ has an Unwinding Family

To prove Lemma 2, arbitrarily fix a state s and data point d. We use proof by induction over j from 0 to t to show that for each pair of states s_1 and s_2 such that $s_1 \mathcal{R}^{2j\epsilon}_{s,d} s_2$, they have the needed properties.

In both the base or inductive cases, since $s_1 \mathcal{R}^{2j\epsilon}_{s,d} s_2$, s_2 must have the same value for the PC as s_1. Thus, they have the same set of enabled actions. That is, there exists a μ_1 such that $s_1 \xrightarrow{a} \mu_1$ iff there exists a μ_2 such that $s_2 \xrightarrow{a} \mu_2$. Thus, $s_1 \overset{a}{\Rightarrow} \nu_1$ iff $s_2 \overset{a}{\Rightarrow} \nu_2$.

Base Case: $j = 0$. For states with a PC of 08, the properties follows from the related states being equal.

For states with a PC of 16, we can prove the needed properties using $\delta = 0$ as we must since $\mathcal{R}^0_{s,d}$ is a 0-unwinding relation. Since $j = 0$ and $s_1 \mathcal{R}^{2j\epsilon}_{s,d} s_2$, s_1 must have the form $\langle 16, \langle B'_0, \ldots, B'_{t-1}\rangle, \langle n_0, \ldots, n'_{t-1}\rangle, c', y', r', k'\rangle$. Since s_1 is related to another state, it must be in S^t_1. Thus, s_1 is reachable in t queries and $c' = c + (t-1)$. Once `curSlot` is updated by line 17, it will roll over to the value of c. Thus, $s_1 \overset{r'}{\Rightarrow} \mathsf{Dirac}(s'_1)$ where $s'_1 = \langle 08, \langle B''_0, \ldots, B''_{t-1}\rangle, \langle n''_0, \ldots, n''_{t-1}\rangle, c, y', r', k'\rangle)$ where $B''_c = \{\!\{\}\!\}$, $n''_c = 0$, and for all $c'' \neq c$, $B''_{c''} = B'_{c''}$ and $n''_{c''} = n'_{c''}$. Since the cth slot was holding the data point by which s_1 and $\mathsf{add}(s_1, c, d)$ differ and s_1 differs from $\mathsf{swap}(s_1, c, d, d')$ for each value of d', $\mathsf{add}(s_1, c, d) \overset{r'}{\Rightarrow} \mathsf{Dirac}(s'_1)$ and $\mathsf{swap}(s_1, c, d, d') \overset{r'}{\Rightarrow} \mathsf{Dirac}(s'_1)$ for all d'. We use β that maps s'_1 to itself and nothing else to anything. Furthermore, for the one state s'_1 in $\mathsf{Supp}(\nu_1)$, $|\ln \nu_1(s'_1) - \ln \nu_2(\beta(s'_1))| = 0 = \delta$. Thus, $\nu_1 \mathcal{L}(=, 0) \nu_2$ where equality is trivially a 0-unwinding relation.

Inductive Case: $j > 0$. We consider cases depending on what type of action a is to show that there exists δ in $[0, 2j\epsilon]$ such that $\mu_1 \mathcal{L}(\mathcal{R}^{2j\epsilon-\delta}_{s,d}, \delta) \mu_2$:

- Subcase: $a \in D$. In this case, we prove that such a δ exists using $\delta = 0$. That is, we prove that $\nu_1 \mathcal{L}(\mathcal{R}^{2j\epsilon}_{s,d}, 0) \nu_2$. Since $a \in D$, s_1 has must have the form

$$\langle 08, \langle B'_0, \ldots, B'_{t-1}\rangle, \langle n'_0, \ldots n'_{t-1}\rangle, c', y', r', k'\rangle$$

  We consider subsubcases:

  - Subsubcase: $c = c'$ and $n_{c'} < v - 1$. In this case, both states s_1 and s_2 will store the data point a. For that c', $\nu_1 = \mathsf{Dirac}(\langle 08, \vec{B}'', \vec{n}'', c', a, r', k'\rangle)$ where $B''_{c'} = B'_{c'} \uplus \{a\}$, $n''_{c'} = n'_{c'} + 1$, and for all $c'' \neq c'$, $B''_{c''} = B'_{c''}$ and $n''_{c''} = n'_{c''}$. Similarly, $\nu_2 = \mathsf{Dirac}(\langle 08, \langle \vec{B}''', \vec{n}''', c', a, r', k'\rangle)$ where either

1. $B'''_{c'} = B_{c'} \uplus \{d\} \uplus \{a\}$, $n'''_{c'} = n'_{c'} + 2$, and for all $c \neq c'' \neq c'$, $B'''_{c''} = B'_{c''}$ and $n'''_{c''} = n'_{c''}$; or
2. $B'''_{c'} = B_{c'} \uplus \{d\} - \{d'\} \uplus \{a\}$, $n'''_{c'} = n'_{c'} + 2$, and for all $c \neq c'' \neq c'$, $B'''_{c''} = B'_{c''}$ and $n'''_{c''} = n'_{c''}$ for some d'.

Thus, for $s'_1 \in \mathsf{Supp}(\nu_1)$ and $s'_2 \in \mathsf{Supp}(\nu_2)$, s'_2 is either $\mathsf{add}(s'_1, c, d)$ or $\mathsf{swap}(s'_1, c, d, d')$ for some d'.

To show that $\mu_1 \ \mathcal{L}(\mathcal{R}^{\epsilon'}_{s,d}, 0) \ \mu_2$, we use the function β that maps s'_1 to the state s'_2 and nothing else. Since both ν_1 and ν_2 are Dirac distributions, that covers all of their supports and is a bijection. It follows from s'_2 being either $\mathsf{add}(s'_1, c, d)$ of $\mathsf{swap}(s'_1, c, d, d')$ for some d' that $s'_1 \ \mathcal{R}^{2j\epsilon}_{s,d} \ s'_2$. Lastly, $|\ln \nu_1(s'_1) - \ln \nu_2(s'_2)| = |\ln 1 - \ln 1| = 0 \leq \delta$

– Subsubcase: $c \neq c'$ and $n_{c'} < v$. Mostly, as above.

– Subsubcase: $n_{c'} = v$. In this case, both states s_1 and s_2 will drop the data point a and not store it. For that c', $\nu_1 = \mathsf{Dirac}(s_1)$ and $\nu_2 = \mathsf{Dirac}(s_2)$ By assumption, $s_1 \ \mathcal{R}^{2j\epsilon}_{s,d} \ s_2$. $|\ln \nu_1(s_1) - \ln \nu_2(s_2)| = |\ln 1 - \ln 1| = 0 \leq \delta$

– Subsubcase: $c = c'$ and $n_{c'} = v - 1$. If $s_2 = \mathsf{swap}(s_1, c, d, d')$ for some d', then this subsubcase is the same as the first one. Otherwise, the s_1 will store the data point, but $s_2 = \mathsf{add}(s_1, c, d)$ will not since it already has $n_{c'} + 1 = v$ data points. Thus, $\nu_2 = \mathsf{Dirac}(s_2)$ and $\nu_1 = \mathsf{Dirac}(s'_1)$ where $s'_1 = \langle 08, \langle B''_0, \ldots, B''_{t-1}\rangle, \langle n''_1, \ldots, n''_{t-1}\rangle, c', a, r', k'\rangle)$ where $B''_{c'} = B'_{c'} \uplus \{a\}$, $n''_{c'} = n'_{c'} + 1$, and for all $c'' \neq c'$, $B''_{c''} = B'_{c''}$ and $n''_{c''} = n'_{c''}$. Thus, we have that $s_2 = \mathsf{swap}(s'_1, c, d, a)$. Thus, $s'_1 \ \mathcal{R}^{2j\epsilon}_{s,d} \ s_2$. We use β that maps s'_1 to s_2 and nothing else. Since $|\ln \nu_1(s_1) - \ln \nu_2(s_2)| = 0$, $\nu_1 \ \mathcal{L}(\mathcal{R}^{2j\epsilon}_{s,d,d'}, 0) \ \nu_2$.

- Subcase: $a \in R$. In this case, we prove that such a δ exists using $\delta = 0$. That is, we prove that $\nu_1 \ \mathcal{L}(\mathcal{R}^{2j\epsilon}_{s,d}, 0) \ \nu_2$.

  Since $a \in R$, s_1 must have the form $\langle 16, \langle B'_0, \ldots, B'_{t-1}\rangle, \langle n_0, \ldots, n'_{t-1}\rangle, c', y', r', k'\rangle$. Thus, $s_1 \stackrel{a}{\Rightarrow} \mathsf{Dirac}(s'_1)$ where

  $$s'_1 = \langle 08, \langle B''_0, \ldots, B''_{t-1}\rangle, \langle n''_0, \ldots, n''_{t-1}\rangle, c' + 1 \mod t, y', r', k'\rangle)$$

  where $B''_{c+1 \mod t} = \{\!\{\}\!\}$, $n''_{c+1 \mod t} = 0$, and for all $c'' \neq c + 1 \mod t$, $B''_{c''} = B'_{c''}$ and $n''_{c''} = n'_{c''}$.

  If $s_2 = \mathsf{add}(s_1, c, d)$, then $s_2 \stackrel{r}{\Rightarrow} \mathsf{Dirac}(s'_2)$ where

  $$s'_2 = \langle 16, \langle B''_0, \ldots, B''_{t-1}\rangle, \langle n''_0, \ldots n''_{t-1}\rangle c' + 1 \mod t, y', r', k'\rangle)$$

  where $B''_{c+1 \mod t} = \{\!\{\}\!\}$, $B''_c = B'_c \uplus \{d\}$, $n''_{c+1 \mod t} = 0$, and for all $c'' \neq c + 1 \mod t$, $B''_{c''} = B'_{c''}$ and $n''_{c''} = n'_{c''}$. Since $j > 0$, $c + (t - j) \mod t \neq c$. Thus, the slot by which s_1 differs from s_2 will remain unchanged, and $s'_1 = \mathsf{add}(s'_2, c, d)$.

  By similar reasoning, if $s_2 = \mathsf{swap}(s_1, c, d, d')$ for some d', $s'_1 = \mathsf{swap}(s'_2, c, d, d')$. Thus, either way, $s'_1 \ \mathcal{R}^{2j\epsilon}_{s,d} \ s'_2$. β that maps the one state of $\mathsf{Supp}(\nu_1)$ to the one state of $\mathsf{Supp}(\nu_2)$ shows that $\nu_1 \ \mathcal{L}(\mathcal{R}^{2j\epsilon}_{s,d}, 0) \ \nu_2$ since $|\ln \mu_1(s'_1) - \ln \mu_2(\mathsf{add}(s'_1, c, d))| = |\ln 1 - \ln 1| = 0 \leq \delta$.

- Subcase: $a \in Q$. In this case, we prove that such a δ exists using $\delta = 2\epsilon$. That is, we prove that $\nu_1 \ \mathcal{L}(\mathcal{R}^{2j\epsilon-2\epsilon}_{s,d}, 2\epsilon) \ \nu_2$. In this case, s_1 has the form $\langle 08, \langle B'_1, \ldots, B'_t\rangle, c', y', r', k'\rangle$. ν_1 is

such that

$$\nu_1(\langle 16, \langle B'_0, \ldots, B'_{t-1}\rangle, \langle n'_0, \ldots n'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) = \Pr\left[\kappa_a\left(\biguplus_{\ell=0}^{t-1} B'_\ell\right) = r''\right]$$

and $\nu_1(s'_1) = 0$ for all other states s'_1. ν_2 is either such that

$$\nu_2(\langle 16, \langle B'_0, \ldots, B'_c \uplus \{d\}, \ldots B'_{t-1}\rangle, \langle n'_0, \ldots n'_c + 1, \ldots n'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \\ = \Pr\left[\kappa_a\left(\biguplus_{\ell=0}^{t-1} B'_\ell \uplus \{d\}\right) = r''\right]$$

or

$$\nu_2(\langle 16, \langle B'_0, \ldots, B'_c \uplus \{d\} - \{d'\}, \ldots B'_{t-1}\rangle, \langle n'_0, \ldots n'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \\ = \Pr\left[\kappa_a\left(\biguplus_{\ell=0}^{t-1} B'_\ell \uplus \{d\} - \{d'\}\right) = r''\right]$$

for some d' and $\nu_1(s'_2) = 0$ for all other states s'_2. Let $\mathcal{B}$ denote which of $\biguplus_{\ell=0}^{t-1} B'_\ell \uplus \{d\}$ and $\biguplus_{\ell=0}^{t-1} B'_\ell \uplus \{d\} - \{d'\}$ it is. Either way $\biguplus_{\ell=0}^{t-1} B'_\ell$ and $\mathcal{B}$ differ by at most two elements Since κ_a has ϵ-differential privacy, we know that for any r'',

$$\Pr\left[\kappa_a\left(\biguplus_{\ell=0}^{t-1} B'_\ell\right) = r''\right] \leq \exp(2\epsilon) * \Pr\left[\kappa_a(\mathcal{B}) = r''\right]$$

Thus,

$$\nu_1(\langle 16, \langle B'_0, \ldots, B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \\ \leq \exp(2\epsilon) * \nu_2(\langle 16, \langle B'_0, \ldots, B'_c \uplus \{d\}, \ldots B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \quad (120)$$

and

$$\nu_1(\langle 16, \langle B'_0, \ldots, B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \\ \leq \exp(2\epsilon) * \nu_2(\langle 16, \langle B'_0, \ldots, B'_c \uplus \{d\} - \{d'\}, \ldots B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \quad (121)$$

Similarly,

$$\nu_2(\langle 16, \langle B'_0, \ldots, B'_c \uplus \{d\}, \ldots B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \\ \leq \exp(2\epsilon) * \nu_1(\langle 16, \langle B'_0, \ldots, B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \quad (122)$$

and

$$\nu_2(\langle 16, \langle B'_0, \ldots, B'_c \uplus \{d\} - \{d'\}, \ldots B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \\ \leq \exp(2\epsilon) * \nu_1(\langle 16, \langle B'_0, \ldots, B'_{t-1}\rangle, c', a, r'', \kappa_a\rangle) \quad (123)$$

```
isInLiftedRelation(S_⊥, R, δ, ν1, ν2)
  V_L := {}
  V_R := {}
  E := {}
  for all x1 ∈ S_⊥
    if ν1(x1) > 0,
      add x1 to V_L
  for all x2 ∈ S_⊥
    if ν2(x2) > 0,
      add x2 to V_R
  for all x1 ∈ V_L
    for all x2 ∈ V_R
      if(x1 R x2 and |ln ν1(x1) − ln ν2(x2)| ≤ δ)
        add edge ⟨x1, x2⟩ to E
  return HopcroftKarpHasPerfectMatching(⟨V_L, V_R, E⟩)
```

Figure 5: Algorithm for checking δ-approximate lifting of relations.

To show that $\nu_1\ \mathcal{L}(\mathcal{R}_{s,d}^{2j\epsilon-2\epsilon}, 2\epsilon)\ \nu_2$, we use a function β. In the case where $s_2 = \mathsf{add}(s_1, c, d)$, β maps each state s_1' of $\mathsf{Supp}(\mu_1)$ to $\mathsf{add}(s_1', c, d)$. To show that β is a bijection from $\mathsf{Supp}(\mu_1)$ to $\mathsf{Supp}(\mu_2)$ note that $\mathsf{add}(\cdot, c, d)$ is a bijection and that Lines 120 and 122 imply that s_1' is in $\mathsf{Supp}(\mu_1)$ iff $\mathsf{add}(s_1', c, d)$ is in $\mathsf{Supp}(\mu_2)$.

In the case where $s_2 = \mathsf{swap}(s_1, c, d, d')$, β maps each s_1' to $\mathsf{swap}(s_1', c, d, d')$. To show that β is a bijection from $\mathsf{Supp}(\mu_1)$ to $\mathsf{Supp}(\mu_2)$ note that $\mathsf{swap}(\cdot, c, d, d')$ is a bijection and that Lines 121 and 123 imply that s_1' is in $\mathsf{Supp}(\mu_1)$ iff $\mathsf{swap}(s_1', c, d, d')$ is in $\mathsf{Supp}(\mu_2)$.

Since $\mathcal{R}_{s,d}^{2(j-1)\epsilon} = \mathcal{R}_{s,d}^{2j\epsilon-2\epsilon}$, for all r'', $s_1'\ \mathcal{R}_{s,d}^{2j\epsilon-2\epsilon}\ \beta(s_1')$. Furthermore, for all s_1' in $\mathsf{Supp}(\mu_1)$, $|\ln \mu_1(s_1') - \ln \mu_2(\beta(s_1'))| \leq \epsilon \leq 2\epsilon \leq \delta$ from Lines 120, 121, 122, and 123.

This completes the proof of the lemma.

Since $\mathcal{R}_{s,d}^{2j*\epsilon}$ covers s and d for all states s and data points d of the automaton M_{ex1}, Lemma 2 and Theorem 2 together prove that the automaton has $(2t * \epsilon)$-differential noninterference.

G The `isInLiftedRelation` Algorithm

The reduction used by `isInLiftedRelation` is shown in Figure 5. First the algorithm constructs the bipartite graph for the reduction and then uses the Hopcroft-Karp algorithm [HK73]. This algorithm returns if and only if there exists a *perfect matching* M for the graph. A perfect matching M for a bipartite graph $\langle V_{\mathsf{L}}, V_{\mathsf{R}}, E\rangle$ is a subset of E such that for every vertex $v \in V = V_{\mathsf{L}} \cup V_{\mathsf{R}}$ is incident to exactly one edge in M.

(Since $\mathsf{Supp}(\nu_1)$ and $\mathsf{Supp}(\nu_2)$ might not be disjoint, but V_{L} and V_{R} must be disjoint, we should tag the states x_1 and x_2 differently before adding them to the sets. However, for readability, we do not explicitly do this tagging.)

Proposition 16. *For all sets S, relations R over S, non-negative reals δ, and distributions ν_1 and ν_2 over S,* `isInLiftedRelation`$(S, \mathsf{R}, \delta, \nu_1, \nu_2)$ *returns true iff $\nu_1\ \mathcal{L}(\mathsf{R}, \delta)\ \nu_2$.*

Proof. By the correctness of the Hopcroft-Karp algorithm, `HopcroftKarpHasPerfectMatching` (and, thus, `isInLiftedRelation`) will only return true if there exists a perfect matching M for the graph.

To prove only-if direction, assume that such an M exists. Given a perfect matching M of bipartite graph, for every $x_1 \in V_{\mathsf{L}}$ there exists a unique edge $e \in E$ such that there exists a $x_2 \in V_{\mathsf{R}}$ such that $e = \langle x_1, x_2 \rangle$. For each such x_1, denote the unique x_2 paired with it by this edge as $\beta_M(x_1)$. β_M is a function from V_{L} to V_{R} since for every $x_1 \in V_{\mathsf{L}}$, there exists exactly one such edge and, thus, exactly one such x_2, which must be in V_{R} since the graph is bipartite. Furthermore, β_M is a bijection since every x_2 in V_{R} must be incident to exactly one edge in the perfect matching M.

Since $V_{\mathsf{L}} = \mathsf{Supp}(\nu_1)$ and $V_{\mathsf{R}} = \mathsf{Supp}(\nu_2)$, β_M is a bijection from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$. Since x_1 and $\beta_M(x_1)$ are connected by an edge, $x_1 \mathsf{R} \beta_M(x_1)$ and $|\ln \nu_1(x_1) - \ln \nu_2(\beta_M(x_1))| \leq \delta$. Thus, the bijection β_M is such that for all $x_1 \in \mathsf{Supp}(\nu_1)$, $x_1 \mathsf{R} \beta_M(x_1)$ and $|\ln \nu_1(x_1) - \ln \nu_2(\beta_M(x_1))| \leq \delta$. This implies that $\nu_1 \mathcal{L}(\mathsf{R}, \delta) \nu_2$.

To prove the if-direction, assume that $\nu_1 \mathcal{L}(\mathsf{R}, \delta) \nu_2$. Then there exists a bijection β from $\mathsf{Supp}(\nu_1)$ to $\mathsf{Supp}(\nu_2)$ such that $x_1 \mathsf{R} \beta(x_1)$ and $|\ln \nu_1(x_1) - \ln \nu_2(\beta(x_1))| \leq \delta$. Let M_β be the set such that $\langle x_1, x_2 \rangle \in M_\beta$ iff $\beta(x_1) = x_2$. M_β is a subset of E since $x_1 \in \mathsf{Supp}(\nu_1)$, $\beta(x_1) \in \mathsf{Supp}(\nu_2)$, $x_1 \mathsf{R} \beta(x_1)$, and $|\ln \nu_1(x_1) - \ln \nu_2(\beta(x_1))| \leq \delta$ together imply that $\langle x_1, \beta(x_1) \rangle$ is in E. M_β is a perfect matching for the graph since β is a bijection from $V_{\mathsf{L}} = \mathsf{Supp}(\nu_1)$ to $V_{\mathsf{R}} = \mathsf{Supp}(\nu_2)$. □

Proposition 17. `isInLiftedRelation` *runs in* $O(|S|^{2.5})$ *time.*

Proof. Given that we know that we never will attempt to add a duplicate element to any of the sets V_{L}, V_{R}, nor E, all the set operations may be done in constant time. Thus, constructing the graph for the reduction operates in $O(|S|^2)$ time. The Hopcroft-Karp perfect matching algorithm operates in $O(\sqrt{v} * e)$ time where v is the number of vertices and e, the number of edges. That is lower than $O(|S|^{2.5})$ since $e \leq v^2$ and $v = |V_{\mathsf{L}}| + |V_{\mathsf{R}}| \leq 2 * |S|$. Thus, the whole algorithm runs in $O(|S|^{2.5})$ time. □

H Proofs for the Checking Algorithm

Proof of Lemma 3: The Soundness of `isUnwindFam`. Here `rel` represents the relation family $\mathcal{R}$ such that $\mathcal{R}^\epsilon$ is equal to $\texttt{rel}[\lfloor \epsilon/\delta \rfloor]$ for ϵ such that $0 \leq \epsilon \leq t * \delta$. If such a family is an unwinding family for transition system, then it is also one for the transition system with all the hidden states have been converted to the same one.

We prove a stronger fact that implies that $\mathcal{R}$ is an $(t * \delta)$-unwinding family for the converted transition system. Namely, we show that the algorithm will only return true if for all ϵ from $[0, t\delta]$, for all x_1 and x_2 in $S_\perp$ such that $\langle x_1, x_2 \rangle \in \texttt{rel}[\lfloor \epsilon/\delta \rfloor]$, for all a in $I \cup R$, there exists ν_1 such that $x_1 \stackrel{a}{\Rightarrow} \nu_1$ iff there exists ν_2 such that $x_2 \stackrel{a}{\Rightarrow} \nu_2$, and when they do exist, either (1) $\nu_1 \mathcal{L}(\texttt{rel}[\lfloor \epsilon/\delta \rfloor] - 0, 0) \nu_2$ or (2) $\nu_1 \mathcal{L}(\texttt{rel}[\lfloor \epsilon/\delta \rfloor - \delta], \delta) \nu_2$. Condition (1) is satisfied if for all $\langle x_1', x_2' \rangle \in \texttt{rel}[\lfloor \epsilon/\delta \rfloor]$, $\nu_1(x_1') = \nu(x_2')$. Condition (2) is satisfied if for all $\langle x_1', x_2' \rangle \in \texttt{rel}[\lfloor \epsilon/\delta \rfloor - \delta]$, $|\ln \nu_1(x_1') - \ln \nu(x_2')| \leq \delta$.

The algorithm will only return true if none of the preceding `return` statements return false. Firstly, it must be the case that $|\texttt{rel}| = t + 1$. Secondly, the outer most `for` loop must finish executing without any of its `return` statements being reached. This will only happen if for all values of i from the length of the array. For each such value, the algorithm examines the relation $\texttt{rel}[i]$, which is the relation used for all values of ϵ in $[0, t\delta]$ such that $\lfloor \epsilon/\delta \rfloor$ is equal to i. Thus,

by considering each value of i, the algorithm examines the intervals $[0, \delta)$, $[\delta, 2\delta)$, and so on up to $[(t-1)\delta, t\delta)$ and finally the point at $t\delta$. Thus, it examines the whole range $[0, t\delta]$ as required by the above condition.

Each of these examinations consists of looking at every pair of $\langle x_1, x_2 \rangle$ in the relation $\texttt{rel}[i]$, and every action a in $I \cup O$. For each such action a and pair, the algorithm first returns false if it is not the case that $x_1 \stackrel{a}{\Rightarrow} \nu_1$ iff $x_2 \stackrel{a}{\Rightarrow} \nu_2$ for some ν_1 and ν_2 since $T[x_1][a]$ is equal to `nil` only in the case where $x_1 \stackrel{a}{\Rightarrow} \nu_1$ for no ν_1 (and likewise for x_2).

If false was not returned, the algorithm checks if it was because ν_1 and ν_2 both exist. If this is not the case, the examination finishes as nothing more must be shown for this state-action pair.

In the case where ν_1 and ν_2 do exist, we know that x_1 and x_2 are actual states and $\perp$ since $T[\perp][a] = \texttt{nil}$ for all a. The examination then continues with the algorithm computing the values of ν_1 and ν_2 such that $x_1 \stackrel{a}{\Rightarrow} \nu_1$ and $x_2 \stackrel{a}{\Rightarrow} \nu_2$ as described above, which is well defined since x_1 and x_2 are actual states.

Next, it checks if $\nu_1 \ \mathcal{L}(\texttt{rel}[i], 0) \ \nu_2$. $\texttt{isInLiftedRelation}(S_\perp, \texttt{rel}[i], 0, \nu_1, \nu_2)$ will return true iff Condition (1) is satisfied. If Condition (1) is satisfied, the examination is complete and algorithm does not return false on this execution of the loop's body.

If Condition (1) is not satisfied, then algorithm next checks to see if Condition (2) holds. For our restricted set of relation families, Condition (2) cannot hold if i is 0 and Condition (1) does not hold. Thus, the next `if` statement. It uses $\texttt{isInLiftedRelation}(S_\perp, \texttt{rel}[i-1], \delta, \nu_1, \nu_2)$ to check if Condition (2) holds. If any pair is not, the algorithm returns false. If Condition (2) is satisfied, the examination is complete, and algorithm does not return false and this execution of the loop.

Thus, each execution of the loop will only complete without returning false if either Condition (1) or Condition (2) holds. As the loop checks all the needed combinations of states and actions, the algorithm will only return true if the stronger fact that implements $\mathcal{R}$ is an unwinding relation is true.

Proof of Lemma 4: The Running Time of `isUnwindFam`. The conversion of all hidden actions to the same one runs in $O(|H| * |S|)$.

The outer most loop runs over the whole length of `rel`. The next loop is over every pair in $\texttt{rel}[i]$ where $\texttt{rel}[i]$ is a binary relation over states. Thus, there are at most $|S|^2$ pairs in $\texttt{rel}[i]$. The next loop is over every action. Thus, the body of this loop will be executed $O(t * |S|^2 * |A|)$ times.

This body consists of four parts. The first is a simple conditional taking constant time. The second computes ν_1 and ν_2. This takes $O(|H| * |S| + |S|^3)$ time. Since the conversion of all hidden actions to the same one takes $|H| = 1$, this is $O(|S|^3)$. The third is a calls `isInLiftedRelation`, which takes $O(|S|^{2.5})$ time. The forth is a conditional and another call to `isInLiftedRelation` on $\texttt{rel}[i-1]$, which takes $O(|S|^{2.5})$ time. Thus, body is $O(|S|^3)$ time and the whole loop is $O(t * |A| * |S|^4)$.

The algorithm whole algorithm run in $O(|H| * |S| + t * |A| * |S|^4)$, which is $O(t * |A| * |S|^4)$ since $|H| \leq |A|$.

Proof of Theorem 4: The Soundness of `isAllCovered`. The algorithm will only return true if none of the preceding `return` statements return false. That is, the outer most `for` loop must finish executing without any of its `return` statements being reached. This will only happen if for every reachable state s and every data point d, either $T[s][d] = \texttt{nil}$ or each of the following is true:

1. $\nu(\bot) = 0$ and $|\texttt{Rels}[s][d]| \neq t = 1$;
2. for all states s' such that $s' \in \mathsf{Supp}(\nu)$, $\langle s, s' \rangle \in \texttt{Rels}[s][d]$; and
3. $\texttt{isUnwindFam}(\langle S, I, O, T \rangle, \texttt{Rels}[s][d], \delta, t)$ returns true

where $s \stackrel{d}{\Rightarrow} \nu$. In the case where $T[s][d] = \texttt{nil}$, the trivial relation family that consists of only empty relations is a $(t * \delta)$-unwinding family for the automaton. In the case where $T[s][d] \neq \texttt{nil}$, the three conditions above imply $\texttt{Rels}[s][d]$ is a $(t * \delta)$-unwinding family for the automaton by using Lemma 3 on the last condition. Either way, there exists a $(t * \delta)$-unwinding family that covers s and d. Thus, the body of the loop will return false unless there exists such an unwinding family.

As the algorithm checks every reachable s for every d, the loop will not terminate without returning false unless the conditions of Theorem 2 holds. Thus, the algorithm only returns true if the automaton has $(t * \delta)$-differential noninterference.

Proof of Theorem 5: The Running Time of `isAllCovered`. Computing the reachable states can be done in time $O(|S|)$.

The outer most loop executes at most $|S|$ times. The next loop executes at most $|D|$ times. In the case where $T[s][d] \neq \texttt{nil}$, the body takes $O(S^3)$ time to compute ν, $O(|S|)$ for the inner loop, and $O(t * |A| * |S|^4)$ time for running the `isUnwindFam` algorithm (Lemma 4). Thus, the body takes $O(t * |A| * |S|^4)$ time and the whole algorithm takes $O(t * |D| * |A| * |S|^5)$ time.