Synthesizing Coupling Proofs of Differential Privacy*

AWS ALBARGHOUTHI, University of Wisconsin–Madison, USA
JUSTIN HSU, University College London, UK

Differential privacy has emerged as a promising probabilistic formulation of privacy, generating intense interest within academia and industry. We present a push-button, automated technique for verifying ε-differential privacy of sophisticated randomized algorithms. We make several conceptual, algorithmic, and practical contributions: (*i*) Inspired by the recent advances on *approximate couplings* and *randomness alignment*, we present a new proof technique called *coupling strategies*, which casts differential privacy proofs as a winning strategy in a game where we have finite privacy resources to expend. (*ii*) To discover a *winning strategy*, we present a constraint-based formulation of the problem as a set of *Horn modulo couplings* (HMC) constraints, a novel combination of first-order Horn clauses and probabilistic constraints. (*iii*) We present a technique for solving HMC constraints by transforming probabilistic constraints into logical constraints with uninterpreted functions. (*iv*) Finally, we implement our technique in the FairSquare verifier and provide the first automated privacy proofs for a number of challenging algorithms from the differential privacy literature, including Report Noisy Max, the Exponential Mechanism, and the Sparse Vector Mechanism.

CCS Concepts: • **Security and privacy → Logic and verification**; • **Theory of computation → Programming logic**;

Additional Key Words and Phrases: Differential Privacy, Synthesis

ACM Reference Format:
Aws Albarghouthi and Justin Hsu. 2018. Synthesizing Coupling Proofs of Differential Privacy. *Proc. ACM Program. Lang.* 2, POPL, Article 58 (January 2018), 30 pages. https://doi.org/10.1145/3158146

1 INTRODUCTION

As more and more personal information is aggregated into massive databases, a central question is how to safely use this data while protecting privacy. *Differential privacy* [Dwork et al. 2006] has recently emerged as one of the most promising formalizations of privacy. Informally, a randomized program is differentially private if on any two input databases differing in a single person's private data, the program's output distributions are *almost* the same; intuitively, a private program shouldn't depend (or reveal) too much on any single individual's record. Differential privacy models privacy quantitatively, by bounding how much the output distribution can change.

Besides generating intense interest in fields like machine learning, theoretical computer science, and security, differential privacy has proven to be a surprisingly fruitful target for formal verification. By now, several verification techniques can *automatically* prove privacy given a lightly annotated program; examples include linear type systems [Gaboardi et al. 2013; Reed and Pierce 2010] and various flavors of dependent types [Barthe et al. 2015; Zhang and Kifer 2017]. For verification,

*The extended version is available at https://arxiv.org/abs/1709.05361.

Authors' addresses: Aws Albarghouthi, University of Wisconsin–Madison, Computer Sciences Department, 1210 West Dayston St. Madison, WI, 53706, USA, aws@cs.wisc.edu; Justin Hsu, University College London, London, UK, email@justinh.su.

2475-1421/2018/1-ART58
https://doi.org/10.1145/3158146

the key feature of differential privacy is *composition*: private computations can be combined into more complex algorithms while automatically satisfying a similar differential privacy guarantee. Composition properties make it easy to design new private algorithms, and also make privacy feasible to verify.

While composition is a powerful tool for proving privacy, it often gives an overly conservative estimate of the privacy level for more sophisticated algorithms. Verification—in particular, automated verification—has been far more challenging for these examples. However, these algorithms are highly important to verify since their proofs are quite subtle. One example, the *Sparse Vector* mechanism [Dwork et al. 2009], has been proposed in slightly different forms at least six separate times throughout the literature. While there were proofs of privacy for each variant, researchers later discovered that only two of the proofs were correct [Lyu et al. 2017]. Another example, the *Report Noisy Max* mechanism [Dwork and Roth 2014], has also suffered from flawed privacy proofs in the past.

1.1 State-of-the-art in Differential Privacy Verification

Recently, researchers have developed techniques to verify private algorithms beyond composition: *approximate couplings* [Barthe et al. 2016b] and *randomness alignment* [Zhang and Kifer 2017].

Approximate Couplings. Approximate couplings are a generalization of *couplings* [Lindvall 2002] from probability theory. Informally, a coupling models two distributions with a single correlated distribution, while an approximate coupling approximately models the two given distributions. Given two output distributions of a program, the existence of a coupling with certain properties can imply target properties about the two output distributions, and about the program itself.

Approximate couplings are a rich abstraction for reasoning about differential privacy, supporting clean, compositional proofs for many examples beyond the reach of the standard composition theorems for differential privacy. Barthe et al. [2016b] first explored approximate couplings for proving differential privacy, building approximate couplings in a relational program logic apRHL where the rule for the random sampling command selects an approximate coupling. While this approach is quite expressive, there are two major drawbacks: (*i*) Constructing proofs requires significant manual effort—existing proofs are formalized in an interactive proof assistant—and selecting the correct couplings requires considerable ingenuity. (*ii*) Like Hoare logic, apRHL does not immediately lend itself to a systematic algorithmic technique for finding invariants, making it a challenging target for automation.

Randomness Alignment. In an independent line of work, Zhang and Kifer [2017] used a technique called *randomness alignment* to prove differential privacy. Informally, a randomness alignment for two distributions is an injection pairing the samples in the first distribution with samples in the second distribution, such that paired samples have approximately the same probability. Zhang and Kifer [2017] implemented their approach in a semi-automated system called LightDP, combining a dependent type system, a custom type-inference algorithm to search for the desired randomness alignments, and a product program construction. Notably, LightDP can analyze standard examples as well as more advanced examples like the Sparse Vector mechanism. While LightDP is an impressive achievement, it, too has a few shortcomings: (*i*) While randomness alignments can be inferred by LightDP's sophisticated type inference algorithm, the full analysis depends on a product program construction with manually-provided loop invariants. (*ii*) Certain examples provable by approximate couplings seem to lie beyond the reach of LightDP. (*iii*) Finally, the analysis in LightDP is technically complex, involving three kinds of program analysis.

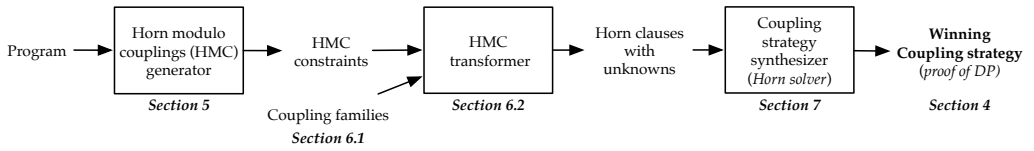


Fig. 1. Overview of our coupling strategy synthesis methodology

While the approaches seem broadly similar, their precise relation remains unclear. Proofs using randomness alignment are often simpler—critical for automation—while approximate couplings seem to be a more expressive proof technique.

1.2 Our Approach: Automated Synthesis of Coupling Proofs

We aim to blend the best features of both approaches—expressivity and automation—enabling fully automatic construction of coupling proofs. To do so, we (*i*) present a novel formulation of proofs of differential privacy via couplings, (*ii*) present a constraint-based technique for discovering coupling proofs, and (*iii*) discuss how to solve these constraints using techniques from automated program synthesis and verification. We describe the key components and novelties of our approach below, and illustrate the overall process in Fig. 1.

Proofs via Coupling Strategies. The most challenging part of a coupling proof is selecting an appropriate coupling for each sampling instruction. Our first insight is that we can formalize the proof technique as discovering a *coupling strategy*, which chooses a coupling for each part of the program while ensuring that the couplings can be composed together. A *winning* coupling strategy proves that the program is differentially private.

To show soundness, we propose a novel, fine-grained version of approximate couplings—*variable approximate couplings*—where the privacy level can vary depending on the pair of states. We show that a winning coupling strategy encodes a variable approximate coupling of the two output distributions, establishing differential privacy for the program. To encode more complex proofs, we give new constructions of variable approximate couplings inspired by the randomness alignments of Zhang and Kifer [2017].

Horn Modulo Couplings. To automatically synthesize coupling strategies, we describe winning coupling strategies as a set of constraints. Constraint-based techniques are a well-studied tool for synthesizing loop invariants, rely-guarantee proofs, ranking functions, etc. For instance, *constrained Horn clauses* are routinely used as a first-order-logic representation of proof rules [Bjørner et al. 2015; Grebenshchikov et al. 2012b]. The constraint-based style of verification cleanly decouples the theoretical task of devising the proof rules from the algorithmic task of solving the constraints.

However, existing constraint systems are largely geared towards deterministic programs—it is not clear how to encode randomized algorithms and the differential privacy property. We present a novel system of constraints called *Horn Modulo Couplings* (HMC), extending Horn clauses with probabilistic *coupling constraints* that use first-order relations to encode approximate couplings. HMC constraints can describe winning coupling strategies.

Solving Horn Modulo Couplings Constraints. Solving HMC constraints is quite challenging, as they combine logical and probabilistic constraints. To simplify the task, we transform probabilistic coupling constraints into logical constraints with unknown expressions. The key idea is that we can restrict coupling strategies to use known approximate couplings from the privacy literature, augmented with a few new constructions we propose in this work.

Our transformation yields a constraint of the form $\exists f.\, \forall x.\, \phi$, read as: there exists a strategy f such that for all inputs x, the program is differentially private. We employ established techniques from synthesis and program verification to solve these simplified constraints. Specifically, we use *counterexample-guided inductive synthesis* (CEGIS) [Solar-Lezama et al. 2006] to discover a strategy f, and *predicate abstraction* [Graf and Saïdi 1997] to prove that f is a winning strategy.

Implementation and Evaluation. We have implemented our technique in the FairSquare probabilistic verification infrastructure [Albarghouthi et al. 2017] and used it to automatically prove ε-differential privacy of a number of algorithms from the literature that have so far eluded automated verification. For example, we give the first fully automated proofs of Report Noisy Max [Dwork and Roth 2014], the discrete exponential mechanism [McSherry and Talwar 2007], and the Sparse Vector mechanism [Lyu et al. 2017].

1.3 Outline and Contributions

After illustrating our technique on a motivating example (§ 2), we present our main contributions.

- We introduce *coupling strategies* as a way of representing coupling proofs of ε-differential privacy (§ 4). To prove soundness, we develop a novel generalization of approximate couplings to support more precise reasoning about the privacy level (§ 3).
- We introduce *Horn Modulo Couplings*, which enrich first-order Horn clauses with probabilistic coupling constraints. We show how to reduce the problem of proving differential privacy to solving Horn Modulo Couplings constraints (§ 5).
- We show how to automatically solve Horn Modulo Couplings constraints by transforming probabilistic coupling constraints into logical constraints with unknown expressions (§ 6).
- We use automated program synthesis and verification techniques to implement our system, and demonstrate its ability to efficiently and automatically prove privacy of a number of differentially private algorithms from the literature (§ 7).

Our approach marries ideas from probabilistic relational logics and type systems with constraint-based verification and synthesis; we compare with other verification techniques (§ 8).

2 MOTIVATION AND ILLUSTRATION

In this section, we demonstrate our verification technique on a concrete example. Our running example is an algorithm from the differential privacy literature called *Report Noisy Max* [Dwork and Roth 2014], which finds the query with the highest value in a list of counting queries. For instance, given a database of medical histories, each query in the list could count the occurrences of a certain medical condition. Then, Report Noisy Max would reveal the condition with approximately the most occurrences, adding random noise to protect the privacy of patients.

We implement this algorithm in Fig. 2 (top) as the program rnm. Given an array q of numeric queries of size $|q|$, rnm evaluates each query, adds noise to the answer, and reports the index of the query with the largest noisy answer. To achieve ε-differential privacy (ε-DP, for short), *Laplacian noise* is added to the result of each query in the first statement of the while loop. (Fig. 2 (bottom) shows two Laplace distributions.) While the code is simple, proving ε-DP is anything but—the only existing formal

```
function rnm(q)
① i, best, r ← 0
② while i < |q|
③     d ~ Lap_{ε/2}(q[i])
④     if d > best ∨ i = 0
⑤         r, best ← i, d
⑥     i ← i + 1
⑦ return r
```

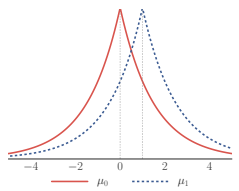


Fig. 2. (top) Report Noisy Max. (bot) Laplace distrs.

proof uses the probabilistic relational program logic apRHL and was carried out in an interactive proof assistant [Barthe et al. 2016b].

2.1 Proof by Approximate Coupling, Intuitively

Differential privacy is a *relational* property of programs (also known as a *hyperproperty*) that compares the output distributions on two different inputs, modeling two hypothetical versions of the private data that should be considered indistinguishable. In the case of rnm, the private data is simply the list of query answers $q[i]$; we will assume that two input states (s_1, s_2) are similar (also called *adjacent*) when each query answer differs by at most 1 in the two inputs; formally, $\forall j \in [0, n).\ |q_1[j] - q_2[j]| \leqslant 1$, where q_1, q_2 are copies of q in the adjacent states, and $|q_1| = |q_2| = n$.

Adding to the verification challenge, differential privacy is also a *quantitative* property, often parameterized by a number ε. To prove ε-differential privacy, we must show that the probability of any final output is approximately the same in *two executions* of the program from any two adjacent inputs. The degree of similarity—and the strength of the privacy guarantee—is governed by ε: smaller values of ε guarantee more similar probabilities, and yields stronger privacy guarantees. Formally, rnm is ε-DP if

for every pair of adjacent inputs q_1 and q_2, every possible output j, and every $\varepsilon > 0$,

$$\Pr[\mathsf{rnm}(q_1) = j] \leqslant \exp(\varepsilon) \cdot \Pr[\mathsf{rnm}(q_2) = j]$$

While it could theoretically be possible to analyze the two executions separately and then verify the inequality, this approach is highly complex. Instead, let us imagine that we step through the two executions side-by-side while tracking the two states. Initially, we have the input states (s_1, s_2). A deterministic instruction simply updates both states, say to (s_1', s_2').

Random sampling instructions $x \sim \mu$ are more challenging to handle. Since we are considering two sampling instructions, in principle we need to consider all possible pairs of samples in the two programs. However, since we eventually want to show that the probability of some output in the first run is close to the probability of that same output in the second run, we can imagine *pairing* each result from the first sampling instruction with a corresponding result from the second sampling instruction, with approximately the same probabilities. This pairing yields a set of paired states; intuitively, one for each possible sample. By flowing this set forward through the program—applying deterministic instructions, selecting how to match samples for sampling instructions, and so on—we end up pairing every possible output state in the first execution with a corresponding output state in the second execution with approximately the same probability.

Proving differential privacy, then, boils down to cleverly finding a pairing for each sampling instruction in order to ensure that all paired output states are related in some particular way. For instance, if whenever the first output state has return value 5 its matching state also has return value 5, then the probability of returning 5 is roughly the same in both distributions.

Approximate Couplings. This idea for proving differential privacy can be formalized as a *proof by approximate coupling*, inspired by the *proof by coupling* technique from probability theory (see, e.g., Lindvall [2002]). An approximate coupling for two probability distributions μ_1, μ_2 is a relation $\Lambda \subseteq S \times S \times \mathbb{R}$ attaching a non-negative real number c to every pair of linked (or *coupled*) states (s_1, s_2). The parameter c bounds how far apart the probability of s_1 in μ_1 is from the probability of s_2 in μ_2: when $c = 0$ the two probabilities must be equal, while a larger c allows greater differences in probabilities. This parameter may grow as our analysis progresses through the program, selecting approximate couplings for the sampling instructions as we go. If we have two states (s_1, s_2) that have probabilities that are bounded by c and we then pair the results from the next sampling instruction (v_1, v_2) with probabilities bounded by c', the resulting paired states will have probabilities bounded

by $c + c'$. We can intuitively think of c' as a *cost* that we need to *pay* in order to pair up the samples (v_1, v_2) from two sampling instructions.

To make this discussion more concrete, suppose we sample from the two Laplace distributions in Fig. 2 (bottom) in the two runs. This figure depicts the probability density functions (PDF) of two different *Laplace distributions* over the integers $\mathbb{Z}$:[1] μ_0 (red; solid) is centered at 0 and μ_1 (blue; dashed) is centered at 1. Then, the relation

$$\Lambda_= = \{(x, x, c) \mid x \in \mathbb{Z}\}$$

is a possible approximate coupling for the two distributions, denoted $\mu_0 \leftrightsquigarrow^{\Lambda_=} \mu_1$, where the cost c depends on the width of the Laplace distributions. The coupling $\Lambda_=$ pairs every sample $x \in \mathbb{Z}$ from μ_0 with the same sample x from μ_1. These samples have different probabilities—since μ_0 and μ_1 are not equal—so the coupling records that the two probabilities are within an $\exp(c)$ multiple of one another.[2]

In general, two distributions have multiple possible approximate couplings. Another valid coupling for μ_0 and μ_1 is

$$\Lambda_{+1} = \{(x, x + 1, 0) \mid x \in \mathbb{Z}\}.$$

This coupling pairs each sample x from μ_0 with the sample $x + 1$ from μ_1. Since μ_1 is just μ_0 shifted by 1, we know that the probabilities of sampling x from μ_0 and $x + 1$ from μ_1 are in fact *equal*. Hence, each pair has cost 0 and we incur no cost from selecting this coupling.

Concrete Example. Let us consider the concrete case of rnm with adjacent inputs $q_1 = [0, 1]$ and $q_2 = [1, 0]$, and focus on one possible return value $j = 0$. To prove the probability bound for output $j = 0$, we need to choose couplings for the sampling instructions such that every pair of linked output states has cost $c \leqslant \varepsilon$, and if the first output is 0, then so is the second output. In other words, the coupled outputs should satisfy the relation $r_1 = 0 \implies r_2 = 0$.

In the first iteration, our analysis arrives at the sampling instruction $d \sim \mathsf{Lap}_{\varepsilon/2}(q[0])$, representing a sample from the Laplace distribution with mean $q[0]$ and *scale* $2/\varepsilon$. Since $q_1[0]$ and $q_2[0]$ are 1 apart, we can couple the values of d_1 and d_2 with the first coupling

$$\Lambda_= = \{(x, x, \varepsilon/2) \mid x \in \mathbb{Z}\}.$$

By paying a cost $\varepsilon/2$, we can assume that $d_1 = d_2$; effectively, applying the coupling $\Lambda_=$ as replaces the probabilistic assignments into a single assignment, where the two processes nondeterministically choose a coupled pair of values from $\Lambda_=$. Therefore, the processes enter the same branch of the conditional and always set r to the same value—in particular, if $r_1 = 0$, then $r_2 = 0$, as desired. Similarly, in the second iteration, we can select the same coupling $\Lambda_=$ and pay $\varepsilon/2$. This brings our total cost to $\varepsilon/2 + \varepsilon/2 = \varepsilon$ and ensures that both processes return the same value $r_1 = r_2$—in particular, if $r_1 = 0$ then $r_2 = 0$, again establishing the requirement for ε-DP for output $j = 0$.

This assignment of a coupling for each iteration is an example of a *coupling strategy*: namely, our simple strategy selects $\Lambda_=$ in every iteration. This strategy also applies for more general inputs, but possibly with a different total cost. For instance if we consider larger inputs, e.g., with $|q| = 3$, we will have to pay $\varepsilon/2$ in three iterations, and therefore we cannot conclude ε-DP. Indeed, this strategy only establishes $\frac{|q|\varepsilon}{2}$-differential privacy in general, as we pay $\varepsilon/2$ in each of the $|q|$ loop iterations. Since larger parameters correspond to looser differential privacy guarantees, this is a weaker property. However, Report Noisy Max is in fact ε-DP for arrays of any length. To prove this stronger guarantee, we need a more sophisticated coupling strategy.

[1] For visualization, we use the continuous Laplace distribution; formally, we assume the discrete Laplace distribution.
[2] When c is small, this is roughly a $(1 + c)$ factor.

We call coupling strategies that establish ε-DP *winning couplings strategies*. Intuitively, the verifier plays a game against the two processes: The processes play by making non-deterministic choices, while the verifier plays by selecting and paying for approximate couplings of sampling instructions, aiming to stay under a "budget" ε.

2.2 Synthesizing Winning Coupling Strategies

Let us consider a different coupling strategy for rnm. Informally, we will focus on one possible output j at a time, and we will only pay non-zero cost in the iteration where r_1 may be set to j. By paying for just a single iteration, our cost will be independent of the number of iterations $|q|$. Consider the following winning coupling strategy:

(1) In loop iterations where $i_1 \neq j$, select the following coupling (known as the *null coupling*):

$$\Lambda_{\varnothing} = \{(x_1, x_2, 0) \mid x_1 - x_2 = q_1[i_1] - q_2[i_2], x_1, x_2 \in \mathbb{Z}\}$$

This coupling incurs 0 privacy cost while ensuring that the two samples remain the same distance apart as $q_1[i_1]$ and $q_2[i_2]$.

(2) In iteration $i_1 = j$, select the following coupling (known as the *shift coupling*):

$$\Lambda_{+1} = \{(x, x+1, \varepsilon) \mid x \in \mathbb{Z}\}$$

Let us explain the idea behind this strategy. In all iterations where $i_1 \neq j$, the processes choose two linked values in $\Lambda_{\varnothing}$ that differ by at most 1 (since $|q_1[i_1] - q_2[i_2]| \leqslant 1$). So the two processes may take different branches of the conditional, and therefore disagree on the values of r_1 and r_2. When $i_1 = j$, we select the shift coupling Λ_{+1} to ensure that $d_2 = d_1 + 1$. Therefore, if r_1 is set to j, then r_2 is set to j; this is because (*i*) $i_1 = i_2$ and (*ii*) d_2 will have to be greater than any of the largest values ($best_2$) encountered by the second execution, forcing the second execution to enter the *then* branch and update r_2. Thus, this coupling strategy ensures that if $r_1 = j$ at the end of the execution, then $r_2 = j$. Since a privacy cost of ε is only incurred for the single iteration when $i_1 = j$, each pair of linked outputs has cost ε. This establishes ε-DP of rnm.

Constraint-Based Formulation of Coupling Proofs. We can view the coupling strategy as assigning a first-order relation

$$\mathit{strat}(\boldsymbol{v}, z_1, z_2, \theta)$$

over (z_1, z_2, θ) to every sampling instruction in the program. The vector of variables $\boldsymbol{v}$ contains two copies the program variables, along with two logical variables representing the output ι we are analyzing and the privacy parameter ε.

Fixing a particular program state $\boldsymbol{c}$, we interpret $\mathit{strat}(\boldsymbol{c}, z_1, z_2, \theta)$ as a ternary relation over $\mathbb{Z}\times\mathbb{Z}\times\mathbb{R}$—a coupling of the two (integer-valued) distributions in the associated sampling instruction, where θ records the cost of this coupling. Effectively, the relation *strat* encodes a function that takes the current state of the two processes $\boldsymbol{v}$ as an input and returns a coupling.

Finding a winning coupling strategy, then, boils down to finding an appropriate interpretation of the relation *strat*. To do so, we generate a system of constraints C whose solutions are winning coupling strategies. A solution of C encodes a proof of ε-DP. For instance, our implementation finds the following solution of *strat* for rnm (simplified for illustration):

$$\bigwedge \begin{array}{ll} i_1 \neq \iota & \implies z_1 - z_2 = q_1[i_1] - q_2[i_2] \wedge \theta = 0 \\ i_1 = \iota & \implies z_2 = z_1 + 1 \wedge \theta = 2 \cdot (\varepsilon/2) \end{array}$$

This is a first-order Boolean formula whose satisfying assignments are elements of *strat*; the two conjuncts encode the null and shift coupling, respectively.

Horn Modulo Couplings. More generally, we work with a novel combination of first-order-logic constraints—in the form of *constrained Horn clauses*—and probabilistic *coupling constraints*. We call our constraints *Horn Modulo Couplings* (HMC) constraints.

Below, we illustrate the more interesting HMC constraints generated by our system for rnm. Suppose that the unknown relation inv_ℓ captures the *coupled invariant* of the two processes at line number ℓ. Just like an invariant for a sequential program encodes the set of reachable states at a program location, the coupled invariant describes the set of reachable states of the two executions (along with privacy cost ω) assuming a particular coupling of the sampling instructions.

$$C_1 : inv_2(\boldsymbol{v}, \omega) \wedge i_1 < |q_1| \wedge i_2 < |q_2| \longrightarrow inv_3(\boldsymbol{v}, \omega)$$
$$C_2 : inv_7(\boldsymbol{v}, \omega) \longrightarrow \omega \leqslant \varepsilon \wedge (r_1 = \iota \Rightarrow r_2 = \iota)$$
$$C_3 : inv_3(\boldsymbol{v}, \omega) \wedge strat(\boldsymbol{v}, z_1, z_2, \theta) \wedge \omega' = \omega + \theta \longrightarrow inv_4(\boldsymbol{v}[d_1 \mapsto z_1, d_2 \mapsto z_2], \omega')$$

Informally, a Horn clause is an implication ($\rightarrow$) describing the flow of states between consecutive lines of the program, and constraints on permissible states at each program location. Clause C_1 encodes the loop entry condition: if both processes satisfy the loop-entry condition at line 2, then both states are propagated to line 3. Clause C_2 encodes the conditions for ε-DP at the end of the program (line 7): the coupled invariant implies that incurred privacy cost ω is at most ε, and if the first process returns ι, then so does the second process. Clause C_3 encodes the effect of selecting the coupling from the strategy at line 3—ω is incremented by the privacy cost θ and d_1, d_2 are updated to new values z_1, z_2, non-deterministically chosen from the coupling provided by the strategy.

Clauses 1–3 are standard constrained Horn clauses, but they are not enough: we need to ensure that *strat* encodes a coupling. So, our constraint system also features probabilistic *coupling constraints*:

$$C_4 : \mathsf{Lap}_{\varepsilon/2}(q_1[i_1]) \leftrightsquigarrow^{strat(\boldsymbol{v},-,-,-)} \mathsf{Lap}_{\varepsilon/2}(q_2[i_2])$$

The constraint says that if we fix the first argument $\boldsymbol{v}$ of *strat* to be the current state, the resulting ternary relation is a coupling for the two distributions $\mathsf{Lap}_{\varepsilon/2}(q_1[i_1])$ and $\mathsf{Lap}_{\varepsilon/2}(q_2[i_2])$.

Solving HMC Constraints. To solve HMC constraints, we transform the coupling constraints into standard Horn clauses with unknown expressions (uninterpreted functions). To sketch the idea, observe that a strategy can be encoded by a formula of the following form:

$$(\varphi_1 \Rightarrow \Psi_1) \wedge \cdots \wedge (\varphi_n \Rightarrow \Psi_n)$$

This describes a *case statement*: if the state of the processes satisfies φ_i, then select coupling Ψ_i. To find formulas φ_i and Ψ_i, we transform solutions of *strat* to a formula of the above form, parameterized by unknown expressions. By syntactically restricting Ψ_i to known couplings, we end up with a system of Horn clauses with unknown expressions in φ_i and Ψ_i whose satisfying assignments always satisfy the coupling constraints.

Finally, solving a system of Horn clauses with unknown expressions is similar to program synthesis for programs with holes, where we want to find a completion that results in a correct program; here, we want to synthesize a coupling strategy proving ε-DP. To implement our system, we leverage automated techniques from program synthesis and program verification.

3 DIFFERENTIAL PRIVACY AND VARIABLE APPROXIMATE COUPLINGS

In this section we define our program model (§ 3.1) and ε-differential privacy (§ 3.2). We then present *variable approximate couplings* (§ 3.3), which generalize the approximate couplings of Barthe et al. [2017b, 2016b] and form the theoretical foundation for our coupling strategies proof method.

3.1 Program Model

Probability Distributions. To model probabilistic computation, we will use probability distributions and sub-distributions. A function $\mu : B \to [0, 1]$ defines a *sub-distribution* over a countable set B if $\sum_{b \in B} \mu(b) \leqslant 1$; when $\sum_{b \in B} \mu(b) = 1$, we call μ a *distribution*. We will often write $\mu(A)$ for a subset $A \subseteq B$ to mean $\sum_{x \in A} \mu(x)$. We write $dist_{\downarrow}(B)$ and $dist(B)$ for the set of all sub-distributions and distributions over B, respectively.

We will use a few standard constructions on sub-distributions. First, the *support* of a sub-distribution μ is defined as $supp(\mu) = \{b \in B \mid \mu(b) > 0\}$. Second, for a sub-distribution on pairs $\mu \in dist_{\downarrow}(B_1 \times B_2)$, the first and second *marginals* of μ, denoted $\pi_1(\mu)$ and $\pi_2(\mu)$, are sub-distributions in $dist_{\downarrow}(B_1)$ and $dist_{\downarrow}(B_2)$:

$$\pi_1(\mu)(b_1) \triangleq \sum_{b_2 \in B_2} \mu(b_1, b_2) \qquad \pi_2(\mu)(b_2) \triangleq \sum_{b_1 \in B_1} \mu(b_1, b_2).$$

We will use $M : W \to dist_{\downarrow}(B)$ to denote a *distribution family*, which is a function mapping every parameter in some set of *distribution parameters* W to a sub-distribution in $dist_{\downarrow}(B)$.

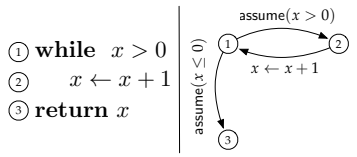
Variables and Expressions. We will work with a simple computational model where programs are labeled graphs. First, we fix a set V of *program variables*, the disjoint union of *input variables* V^i and *local variables* V^l. Input variables model inputs to the program and are never modified—they can also be viewed as logical variables. We assume that a special, real-valued input variable ε contains the target privacy level. The set of local variables includes an *output variable* $v_r \in V^l$, representing the value returned by the program. We assume that each variable v has a type D_v, e.g., the Booleans $\mathbb{B}$, the natural numbers $\mathbb{N}$, or the integers $\mathbb{Z}$.

We will consider several kinds of expressions, built out of variables. We will fix a collection of primitive operations; for instance, arithmetic operations + and ·, Boolean operations =, ∧, ∨, etc. We shall use *exp* to denote expressions over variables V, *bexp* to denote Boolean expressions over V, and *iexp* to denote input expressions over V^i. We consider a set of distribution expressions *dexp* modeling primitive, built-in distributions. For our purposes, distribution expressions will model standard distributions used in the differential privacy literature; we will detail these when we introduce differential privacy below. Distribution expressions can be parameterized by standard expressions.

Finally, we will need three kinds of basic program statements $\mathfrak{st}$. An *assignment statement* $v \leftarrow exp$ stores the value of *exp* into the variable v. A *sampling statement* $v \sim dexp$ takes a fresh sample from the distribution *dexp* and stores it into v. Finally, an *assume statement* assume(*bexp*) does nothing if *bexp* is true, otherwise it filters out the execution (fails to terminate).

Programs. A program $P = (L, E)$ is a directed graph where nodes L represent program locations, directed edges $E \subseteq L \times L$ connect pairs of locations, and each edge $e \in E$ is labeled by a statement $\mathfrak{st}_e$. There is a distinguished *entry location* $\ell_{en} \in L$, representing the first instruction of the program, and a *return location* $\ell_{ret} \in L$. We assume all locations in L can reach ℓ_{ret} by following edges in E.

Our program model is expressive enough to encode the usual conditionals and loops from imperative languages—e.g., consider the program on the right along with its graph representation. All programs we consider model structured programs, e.g., there is no control-flow non-determinism.



Expression Semantics. A program *state* s is a map assigning a value to each variable in V; let S denote the set of all possible states. Given variable v, we use $s(v)$ to denote the value of v in state s. Given constant c, we use $s[v \mapsto c]$ to denote the state s but with variable v mapped to c. The

semantics of an expression *exp* is a function $[\![exp]\!] : S \to D$ from a state to an element of some type D. For instance, the expression $x + y$ in state s is interpreted as $[\![x + y]\!](s) = [\![x]\!](s) + [\![y]\!](s)$. A distribution expression *dexp* is interpreted as a distribution family $[\![dexp]\!] : S \to dist(\mathbb{Z})$, mapping a state in S to a distribution over integers; for concreteness we will only consider distributions over integers, but extending to arbitrary discrete distributions is straightforward. We will often write $s(-)$ to denote $[\![-]\!](s)$, for simplicity.

Program Semantics. We can define semantics of a program P as the aggregate of all its traces. A *trace* σ through P is a finite sequence of statements $\mathfrak{st}_{e_1}, \ldots, \mathfrak{st}_{e_n}$ such that the associated edges edges $e_1, \ldots, e_n \in E$ form a directed path through the graph. A *maximal trace* σ has edges corresponding to a path from ℓ_{en} to ℓ_{ret}. We use $\Sigma(P)$ to denote the set of all maximal traces through P. A trace σ represents a sequence of instructions to be executed, so we define $[\![\sigma]\!] : S \to dist_{\downarrow}(S)$ as a distribution family from states to sub-distributions over states. Assignments and sampling statements are given the expected semantics, and an assume(*bexp*) statement yields the all-zeros sub-distribution if the input state does not satisfy the guard *bexp*. We can now define the semantics of a full program $[\![P]\!] : S \to dist(S)$ as

$$[\![P]\!](s) \triangleq \sum_{\sigma \in \Sigma(P)} [\![\sigma]\!](s)$$

where the sum adds up output sub-distributions from each trace. We will assume P is terminating; in particular, the sub-distributions $[\![\sigma]\!](s)$ sum to a proper distribution.

3.2 Differential Privacy

Differential privacy is a quantitative, statistical notion of data privacy proposed by Dwork et al. [2006]; we reformulate their definition in our program model. We will fix an *adjacency relation* on input states $\Delta \subseteq S \times S$ modeling which inputs should lead to similar outputs. Throughout, we implicitly assume (*i*) for all $(s_1, s_2) \in \Delta$, $s_1(\varepsilon) = s_2(\varepsilon) > 0$, and (*ii*) for all $(s_1, s_2) \in \Delta$ and every $c \in \mathbb{R}^{>0}$, $(s_1[\varepsilon \mapsto c], s_2[\varepsilon \mapsto c]) \in \Delta$. In other words, in adjacent states ε may take any positive value, as long as it is equal in both states.

Definition 3.1 (Dwork et al. [2006]). A program P is *ε-differentially private* with respect to an adjacency relation Δ iff for every $(s_1, s_2) \in \Delta$ and output $j \in D_{v_r}$,

$$\mu_1(\{s \mid s(v_r) = j\}) \leqslant \exp(c) \cdot \mu_2(\{s \mid s(v_r) = j\})$$

where $\mu_1 = P(s_1)$, $\mu_2 = P(s_2)$, and $s_1(\varepsilon) = s_2(\varepsilon) = c$. ■

Dwork et al. [2006] originally defined differential privacy in terms of *sets* of outputs rather than single outputs. Def. 3.1 is equivalent to their notion of $(\varepsilon, 0)$-differential privacy and is more convenient for our purposes, as we will often prove privacy by focusing on one output at at time.

We will consider two primitive distributions that are commonly used in differentially private algorithms: the (discrete) *Laplace* and *exponential* distributions. Both distributions are parameterized by two numbers, describing the spread of the distribution and its center.

Definition 3.2. The *Laplace distribution* family is a function with two parameters, $y \in \mathbb{R}^{>0}$ and $z \in \mathbb{Z}$. We use $\mathsf{Lap}_y(z)$ to denote the distribution over $\mathbb{Z}$ where $\mathsf{Lap}_y(z)(v) \propto \exp(-|v - z| \cdot y)$, for all $v \in \mathbb{Z}$. We call $1/y$ the *scale* of the distribution and z the *mean* of the distribution. ■

Definition 3.3. The *exponential distribution* family is a function with two parameters, $y \in \mathbb{R}^{>0}$ and $z \in \mathbb{Z}$. We use $\mathsf{Exp}_y(z)$ to denote the distribution over $\mathbb{Z}$ where $\mathsf{Exp}_y(z)(v) = 0$ for all $v < z$, and $\mathsf{Exp}_y(z)(v) \propto \exp(-(v - z) \cdot y)$ for all $v \geqslant z$. ■

We will use distribution expressions $\mathrm{Lap}_{iexp}(exp)$ and $\mathrm{Exp}_{iexp}(exp)$ to represent the respective Laplace/exponential distribution interpreted in a given state. We assume that every pair of adjacent states $(s_1, s_2) \in \Delta$ satisfies $s_1(iexp) = s_2(iexp) > 0$ for every *iexp* appearing in distribution expressions.

3.3 Variable Approximate Couplings

We are now ready to define the concept of a *variable approximate coupling*, which we will use to establish ε-DP of a program. A variable approximate coupling of two distributions μ_1 and μ_2 can be viewed as two distributions on pairs, each modeling one of the original distributions. Formally:

Definition 3.4 (Variable approximate couplings). Let $\mu_1 \in dist_{\downarrow}(B_1)$ and $\mu_2 \in dist_{\downarrow}(B_2)$. Let $\Lambda \subseteq B_1 \times B_2 \times \mathbb{R}^{\geqslant 0}$ be a relation. We write $dom(\Lambda) \subseteq B_1 \times B_2$ for the projection of Λ to the first two components. We say that Λ is a *variable approximate coupling* of μ_1 and μ_2—denoted $\mu_1 \leftrightsquigarrow^{\Lambda} \mu_2$—if there exist two *witness* sub-distributions $\mu_L \in dist_{\downarrow}(B_1 \times B_2)$ and $\mu_R \in dist_{\downarrow}(B_1 \times B_2)$ such that:

(1) $\pi_1(\mu_L) = \mu_1$ and $\pi_2(\mu_R) \leqslant \mu_2$,
(2) $supp(\mu_L), supp(\mu_R) \subseteq dom(\Lambda)$, and
(3) for all $(b_1, b_2, c) \in \Lambda$, we have $\mu_L(b_1, b_2) \leqslant \exp(c) \cdot \mu_R(b_1, b_2)$.

We will call these the *marginal* conditions, the *support* conditions, and the *distance* conditions respectively. We will often abbreviate "variable approximate coupling" as simply "coupling". ■

To give some intuition, the first point states that μ_L models the first distribution and μ_R is a lower bound on the second distribution—the inequality $\pi_2(\mu_R) \leqslant \mu_2$ means that $\pi_2(\mu_R)(b_2) \leqslant \mu_2(b_2)$ for every $b_2 \in B_2$. Informally, since the definition of differential privacy (Def. 3.1) is asymmetric, it is enough to show that the first distribution (which we will model by μ_L) is less than a lower bound of the second distribution (which we will model by μ_R). The second point states that every pair of elements with non-zero probability in μ_L or μ_R must have at least one cost. Finally, the third point states that μ_L is approximately upper bounded by μ_R; the approximation is determined by a cost c at each pair.

Our definition is a richer version of $\star$-lifting, an approximate coupling recently proposed by Barthe et al. [2017b] for verifying differential privacy. The main difference is our approximation level c may vary over the pairs (b_1, b_2), hence we call our approximate coupling a *variable* approximate coupling. When all approximation levels are the same, our definition recovers the existing definition of $\star$-lifting.[3] Variable approximate couplings can give more precise bounds when comparing events in the first distribution to events in the second distribution.

Now, ε-DP holds if we can find a coupling of the output distributions from adjacent inputs.

Lemma 3.5 (ε-DP and couplings). *A program P is ε-DP with respect to adjacency relation Δ if for every $(s_1, s_2) \in \Delta$ and $j \in D_{v_r}$, there is a coupling $P(s_1) \leftrightsquigarrow^{\Lambda_j} P(s_2)$ with*

$$\Lambda_j = \{(s_1', s_2', c) \mid s_1'(v_r) = j \Rightarrow s_2'(v_r) = j\},$$

where $s_1(\varepsilon) = s_2(\varepsilon) = c$.

We close this section with some examples of specific couplings for the Laplace and exponential distributions proposed by Barthe et al. [2016b].

Definition 3.6 (Shift coupling). Let $y \in \mathbb{R}^{>0}$, $z_1, z_2 \in \mathbb{Z}$, and $k \in \mathbb{Z}$, and define the relations

$$\Lambda_{+k} \triangleq \{(n_1, n_2, |k+z_1-z_2|\cdot y) \mid n_1+k = n_2\} \quad \text{and} \quad \Lambda'_{+k} \triangleq \{(n_1, n_2, (k+z_1-z_2)\cdot y) \mid n_1+k = n_2\}$$

[3] More precisely, $(\varepsilon, 0)$ $\star$-lifting.

Then we have the following *shift* coupling for the Laplace distribution:

$$\mathsf{Lap}_y(z_1) \leftrightsquigarrow^{\Lambda_{+k}} \mathsf{Lap}_y(z_2).$$

If $k + z_1 - z_2 \geqslant 0$, we have the following shift coupling for the exponential distribution:

$$\mathsf{Exp}_y(z_1) \leftrightsquigarrow^{\Lambda'_{+k}} \mathsf{Exp}_y(z_2).$$

Intuitively, these couplings relate each sample n_1 from the first distribution with the sample $n_2 = n_1 + k$ from the second distribution. The difference in probabilities for these two samples depends on the shift k and the difference between the means z_1, z_2. ■

If we set $k = z_2 - z_1$ above, the coupling cost is 0 and we have the following useful special case.

Definition 3.7 (Null coupling). Let $y \in \mathbb{R}^{>0}$, $z_1, z_2 \in \mathbb{Z}$, and define the relation:

$$\Lambda_\varnothing \triangleq \{(n_1, n_2, 0) \mid n_1 - z_1 = n_2 - z_2\}.$$

Then we have the following *null* couplings:

$$\mathsf{Lap}_y(z_1) \leftrightsquigarrow^{\Lambda_\varnothing} \mathsf{Lap}_y(z_2) \quad \text{and} \quad \mathsf{Exp}_y(z_1) \leftrightsquigarrow^{\Lambda_\varnothing} \mathsf{Exp}_y(z_2).$$

Intuitively, these couplings relate pairs of samples n_1, n_2 that are at the same distance from their respective means z_1, z_2—the approximation level is 0 since linked samples have the same probabilities under their respective distributions. ■

4 COUPLING STRATEGIES

In this section, we introduce *coupling strategies*, our proof technique for establishing ε-DP.

4.1 Formalizing Coupling Strategies

Roughly speaking, a coupling strategy picks a coupling for each sampling statement.

Definition 4.1 (Coupling strategies). Let *Stmts* be the set of all sampling statements in a program P; recall that our primitive distributions are over the integers $\mathbb{Z}$. A *coupling strategy* τ for P is a map from $\mathit{Stmts} \times S \times S$ to couplings in $2^{\mathbb{Z}\times\mathbb{Z}\times\mathbb{R}}$, such that for a statement $\mathtt{st} = v \sim \mathit{dexp}$ and states (s_1, s_2), the relation $\tau(\mathtt{st}, s_1, s_2) \subseteq \mathbb{Z} \times \mathbb{Z} \times \mathbb{R}$ forms a coupling $s_1(\mathit{dexp}) \leftrightsquigarrow^{\tau(\mathtt{st}, s_1, s_2)} s_2(\mathit{dexp})$. ■

Example 4.2. Recall our simple coupling strategy for Report Noisy Max in § 2.1. For the statement $\mathtt{st} = v \sim \mathsf{Lap}_{\varepsilon/2}(q[i])$, and every pair of states (s_1, s_2) where $s_1(\varepsilon) = s_2(\varepsilon) = c > 0$, the strategy τ returned the coupling $\tau(\mathtt{st}, s_1, s_2) = \{(x, x, c) \mid x \in \mathbb{Z}\}$. ■

To describe the effect of a coupling strategy on two executions from neighboring inputs, we define a *coupled postcondition* operation. This operation propagates a set of pairs of coupled states while tracking the privacy cost from couplings selected along the execution path.

Definition 4.3 (Coupled postcondition). Let τ be a coupling strategy for P. We define the *coupled postcondition* as a function post_τ, mapping $Q \subseteq S \times S \times \mathbb{R}$ and a statement to a subset of $S \times S \times \mathbb{R}$:

$$\begin{aligned}
\mathit{post}_\tau(Q, v \leftarrow \mathit{exp}) &\triangleq \{(s_1[v \mapsto s_1(\mathit{exp})], s_2[v \mapsto s_2(\mathit{exp})], c) \mid (s_1, s_2, c) \in Q\} \\
\mathit{post}_\tau(Q, \mathsf{assume}(\mathit{bexp})) &\triangleq \{(s_1, s_2, c) \mid (s_1, s_2, c) \in Q \wedge s_1(\mathit{bexp}) \wedge s_2(\mathit{bexp})\} \\
\mathit{post}_\tau(Q, v \sim \mathit{dexp}) &\triangleq \{(s_1[v \mapsto a_1], s_2[v \mapsto a_2], c + c') \\
&\qquad \mid (s_1, s_2, c) \in Q \wedge (a_1, a_2, c') \in \tau(v \sim \mathit{dexp}, s_1, s_2)\}.
\end{aligned}$$

This operation can be lifted to operate on sequences of statements σ in the standard way: when the trace $\sigma = \mathtt{st}_1\sigma'$ begins with $\mathtt{st}_1$, we define $\mathit{post}_\tau(Q, \sigma) \triangleq \mathit{post}_\tau(\mathit{post}_\tau(Q, \mathtt{st}_1), \sigma')$; when σ is the empty trace, $\mathit{post}_\tau(Q, \sigma) \triangleq Q$. ■

Intuitively, on assignment statements *post* updates every pair of states (s_1, s_2) using the semantics of assignment; on assume statements, *post* only propagates pairs of states satisfying *bexp*; on sampling statements, *post* updates every pair of states (s_1, s_2) by assigning the variables (v_1, v_2) with every possible pair from the coupling chosen by τ, updating the incurred privacy costs.

Example 4.4. Consider a simple program:

$x \leftarrow x + 10$

$x \sim \mathsf{Lap}_\varepsilon(x)$

Let the adjacency relation $\Delta = \{(s_1, s_2) \mid s_1(x) = s_2(x) + 1 \text{ and } s_1(\varepsilon) = s_2(\varepsilon) = c\}$ and let τ be a coupling strategy. Since there is only a single variable x and ε is fixed to c, we will represent the states of the two processes by x_1 and x_2, respectively. Then, the initial set of coupled states Q_0 is the set $\{(x_1, x_2, 0) \mid x_1 = x_2 + 1\}$. First, we compute $Q_1 = post_\tau(Q_0, x \leftarrow x + 10)$. This results in a set $Q_1 = Q_0$, since adding 10 to both variables does not change the fact that $x_1 = x_2 + 1$. Next, we compute $Q_2 = post_\tau(Q_1, x \sim dexp)$. Suppose that the coupling strategy τ maps every pair of states $(s_1, s_2) \in Q_1$ to the coupling $\Lambda_= = \{(k, k, c) \mid k \in \mathbb{Z}\}$. Then, $Q_2 = \{(x_1, x_2, c) \mid x_1 = x_2\}$. ■

While a coupling strategy restricts the pairs of executions we must consider, the executions may still behave quite differently—for instance, they may take different paths at conditional statements or run for different numbers of loop iterations, etc. To further simplify the verification task we will focus on *synchronizing* strategies only, which ensure that the guard in every assume instruction takes the same value on coupled pairs, so both coupled processes follow the same control-flow path through the program.

Definition 4.5 (Synchronizing coupling strategies). Given a trace σ, let σ_i denote the prefix $\not{s}_1, \ldots, \not{s}_i$ of σ and let σ_0 denote the empty trace that simply returns the input state. A coupling strategy τ is *synchronizing* for σ and $(s_1, s_2) \in \Delta$ iff for every $\not{s}_i = \text{assume}(bexp)$ and $(s_1', s_2', -) \in post_\tau(\{(s_1, s_2, 0)\}, \sigma_{i-1})$, we have $s_1'(bexp) = s_2'(bexp)$. ■

Requiring synchronizing coupling strategies is a certainly a restriction, but not a serious one for proving differential privacy. By treating conditional statements as monolithic instructions, the only potential branching we need to consider comes from loops. Differentially private algorithms use straightforward iteration; all examples we are aware of can be encoded with for-loops and analyzed synchronously. (§ 7 provides more details about how we handle branching.)

4.2 Proving Differential Privacy via Coupling Strategies

Our main theoretical result shows that synchronizing coupling strategies encode couplings.

Lemma 4.6 (From strategies to couplings). *Suppose τ is synchronizing for σ and $(s_1, s_2) \in \Delta$. Let $\Psi = post_\tau(\{(s_1, s_2, 0)\}, \sigma)$, and let $f : S \times S \to \mathbb{R}$ be such that $c \leqslant f(s_1', s_2')$ for all $(s_1', s_2', c) \in \Psi$. Then we have a coupling $\sigma(s_1) \leftrightsquigarrow^{\Psi_f} \sigma(s_2)$, where $\Psi_f \triangleq \{(s_1', s_2', f(s_1', s_2')) \mid (s_1', s_2', -) \in \Psi\}$.*

Proof. (*Sketch*) By induction on the length of the trace σ. ■

There may be multiple sampling choices that yield the same coupled outputs (s_1', s_2'); the coupling must assign a cost larger than all associated costs in the coupled post-condition. By Lem. 3.5, if we can find a coupling for each possible output $j \in D_{v_r}$ such that $s_1(v_r) = j \implies s_2(v_r) = j$ with cost at most ε, then we establish ε-differential privacy. We formalize this family of coupling strategies as a *winning coupling strategy*.

Theorem 4.7 (Winning coupling strategies). *Fix program P and adjacency relation Δ. Suppose that we have a family of coupling strategies $\{\tau_j\}_{j \in D_{v_r}}$ such that for every trace $\sigma \in \Sigma(P)$ and $j \in D_{v_r}$,*

τ_j is synchronizing for σ and

$$post_{\tau_j}(\Delta \times \{0\}, \sigma) \subseteq \{(s_1', s_2', c) \mid c \leqslant s_1(\varepsilon) \wedge s_1'(v_r) = j \Rightarrow s_2'(v_r) = j\}.$$

Then P is ε-differentially private. (We could change $s_1(\varepsilon)$ to $s_2(\varepsilon)$ above, since $s_1(\varepsilon) = s_2(\varepsilon)$ in Δ.)

Proof. Let $(s_1, s_2) \in \Delta$ be two input states such that $s_1(\varepsilon) = s_2(\varepsilon) = c$. For every $j \in D_{v_r}$ and any sequence $\sigma \in \Sigma(P)$, Lem. 4.6 gives a coupling of the output distributions: $\sigma(s_1) \leftrightsquigarrow^{\Lambda} \sigma(s_2)$ where $s_1'(v_r) = j \implies s_2'(v_r) = j$ and $c' \leqslant c$ for every $(s_1', s_2', c') \in \Lambda$. By a similar argument as in Lem. 3.5, we have

$$\sigma(s_1)(\{s_1' \mid s_1'(v_r) = j\}) \leqslant \exp(c) \cdot \sigma(s_2)(\{s_2' \mid s_2'(v_r) = j\}).$$

Since $[\![P]\!] = \sum_{\sigma \in \Sigma(P)} [\![\sigma]\!]$, summing over all traces $\sigma \in \Sigma(P)$ shows ε-differential privacy:

$$P(s_1)(\{s_1' \mid s_1'(v_r) = j\}) \leqslant \exp(c) \cdot P(s_2)(\{s_2' \mid s_2'(v_r) = j\}).$$ ∎

5 HORN CLAUSES MODULO COUPLINGS

So far, we have seen how to prove differential privacy by finding a winning coupling strategy. In this section we present a constraint-based formulation of winning coupling strategies, paving the way to automation.

5.1 Enriching Horn Clauses with Coupling Constraints

Constrained Horn clauses are a standard tool for logically encoding verification proof rules [Bjørner et al. 2015; Grebenshchikov et al. 2012b]. For example, a set of Horn clauses can describe a loop invariant, a rely-guarantee proof, an Owicki-Gries proof, a ranking function, etc. Since our proofs—winning coupling strategies—describe probabilistic couplings, our Horn clauses will involve a new kind of probabilistic constraint.

Horn Clauses. We assume *formulas* are interpreted in some first-order theory (e.g., linear integer arithmetic) with a set R of uninterpreted *relation* symbols. A *constrained Horn clause* C, or Horn clause for short, is a first-order logic formula of the form

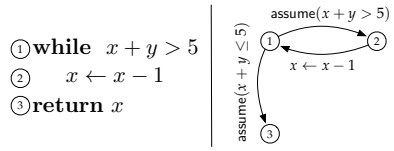
$$r_1(\boldsymbol{v}_1) \wedge r_2(\boldsymbol{v}_2) \wedge \ldots \wedge r_{n-1}(\boldsymbol{v}_{n-1}) \wedge \varphi \longrightarrow H_C$$

where:

- each relation $r_i \in R$ is of arity equal to the length of the vector of variables $\boldsymbol{v}_i$;
- φ is an interpreted formula over the first-order theory (e.g., $x > 0$);
- the left-hand side of the implication ($\longrightarrow$) is called the *body* of C; and
- H_C, the *head* of C, is either a relation application $r_n(\boldsymbol{v}_n)$ or an interpreted formula φ'.

We will allow interpreted formulas to contain disjunctions. For clarity of presentation, we will use $\Rightarrow$ to denote implications in an interpreted formula, and $\rightarrow$ to denote the implication in a Horn clause. All free variables are assumed to be universally quantified, e.g., $x + y > 0 \longrightarrow r(x, y)$ means $\forall x, y.\, x + y > 0 \longrightarrow r(x, y)$. For conciseness we often write $r(\boldsymbol{v}, x_1, \ldots, x_m)$ to denote the relation application $r(v_1, \ldots, v_n, x_1, \ldots, x_m)$, where $\boldsymbol{v}$ is the vector of variables $v_1, \ldots, v_n$.

Semantics. We will write $\mathcal{C}$ for a set of clauses $\{C_1, \ldots, C_n\}$. Let G be a graph over relation symbols such that there is an edge (r_1, r_2) iff r_1 appears in the body of some clause C_i and r_2 appears in its head. We say that $\mathcal{C}$ is *recursive* iff G has a cycle. The set $\mathcal{C}$ is *satisfiable* if there exists an interpretation ρ of relation symbols as relations in the theory such that every clause $C \in \mathcal{C}$ is valid. We say that ρ satisfies $\mathcal{C}$ (denoted $\rho \models \mathcal{C}$) iff for all $C \in \mathcal{C}$, ρC is valid (i.e., equivalent to *true*), where ρC is C with every relation application $r(\boldsymbol{v})$ replaced by $\rho(r(\boldsymbol{v}))$.



$$
\begin{array}{ll}
C_1 : x = y \longrightarrow r_1(x, y) & \text{precondition} \\
C_2 : r_1(x, y) \wedge x + y > 5 \longrightarrow r_2(x, y) & \text{loop entry} \\
C_3 : r_2(x, y) \wedge x' = x - 1 \longrightarrow r_1(x', y) & \text{loop body} \\
C_4 : r_1(x, y) \wedge x + y \leqslant 5 \longrightarrow r_3(x, y) & \text{loop exit} \\
C_5 : r_3(x, y) \longrightarrow x \leqslant y & \text{postcondition}
\end{array}
$$

Fig. 3. Example of using Horn clauses for proving a Hoare triple

The next example uses Horn clauses to prove a simple Hoare triple for a standard, deterministic program. For a detailed exposition of Horn clauses in verification, we refer to the excellent survey by Bjørner et al. [2015].

Example 5.1. Consider the simple program P and its graph representation in Fig. 3. Suppose we want to prove the Hoare triple $\{x = y\}\ P\ \{x \leqslant y\}$. We generate the Horn clauses shown in Fig. 3, where relations r_i denote the Hoare-style annotation at location i of P. Each clause encodes either the pre/postcondition or the semantics of one of the program statements.

Notice that the constraint system is recursive. Intuitively, r_i captures all reachable states at location i; relation r_1 encodes an inductive loop invariant that holds at the loop head. We can give a straightforward reading of each clause. For instance, C_3 states that if (x, y) is reachable at location 2, then $(x - 1, y)$ must be reachable at location 1; C_5 stipulates that all states reachable at location 5 must be such that $x \leqslant y$, i.e., satisfy the postcondition.

One possible satisfying assignment ρ to these constraints is $\rho(r_i) = \{(a, b) \mid a \leqslant b\}$ for $i \in [1, 3]$. Applying ρ to $r_1(x, y)$ results in $x \leqslant y$, the loop invariant. We can interpret C_1 and C_3 under ρ:

$$\rho C_1 : x = y \longrightarrow x \leqslant y \qquad\qquad \rho C_3 : x \leqslant y \wedge x' = x - 1 \longrightarrow x' \leqslant y$$

Observe that all ρC_i are valid, establishing validity of the original Hoare triple. ■

Horn Clauses Modulo Coupling Constraints. We now enrich Horn clauses with a new form of constraint that describes couplings between pairs of distributions; we call the resulting constraints *Horn Modulo Couplings* (HMC).

Definition 5.2 (Coupling constraints). Suppose we have a relation r of arity $n + 3$ for some $n \geqslant 0$, and suppose we have two distribution families M_1 and M_2. A coupling constraint is of the form:

$$M(\boldsymbol{v}_1) \leftrightsquigarrow^{r(\boldsymbol{v}_3, -, -, -)} M(\boldsymbol{v}_2)$$

An interpretation ρ satisfies such a constraint iff for every consistent assignment $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3$ to the variables $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$, we have $M(\boldsymbol{c}_1) \leftrightsquigarrow^{\rho(r(\boldsymbol{c}_3, -, -, -))} M(\boldsymbol{c}_2)$, where $\rho(r(\boldsymbol{c}_3, -, -, -))$ is defined as $\{(k_1, k_2, \theta) \mid (\boldsymbol{c}_3, k_1, k_2, \theta) \in \rho(r)\}$. In other words, $\rho(r(\boldsymbol{c}_3, -, -, -))$ is the ternary relation resulting from fixing the first n components of $\rho(r)$ to $\boldsymbol{c}_3$. Note that the vectors $\{\boldsymbol{v}_i\}_i$ may share variables. ■

Example 5.3. Consider the simple coupling constraint $\mathsf{Lap}_c(x_1) \leftrightsquigarrow^{r(x_1, x_2, -, -, -)} \mathsf{Lap}_c(x_2)$, where c is a positive constant. A simple satisfying assignment is the null coupling from Def. 3.7:

$$\rho(r) = \{(a_1, a_2, b_1, b_2, 0) \mid b_1 - b_2 = a_1 - a_2\}$$

Effectively, $\rho(r)$ says: *for any values* (a_1, a_2) *of* (x_1, x_2)*, the relation* $\Lambda_\varnothing = \{(b_1, b_2, 0) \mid b_1 - b_2 = a_1 - a_2\}$ *is a coupling such that* $\mathsf{Lap}_c(a_1) \leftrightsquigarrow^{\Lambda_\varnothing} \mathsf{Lap}_c(a_2)$. ■

Example 5.4. For a simpler example, consider the constraint $\mathsf{Lap}_c(x) \leftrightsquigarrow^{r(-, -, -)} \mathsf{Lap}_c(x)$, where c is a positive constant. A satisfying assignment to this constraint is: $\rho(r) = \{(b_1, b_2, 0) \mid b_1 = b_2\}$, since the two distributions are the same for any value of variable x. Another possible satisfying assignment is $\rho(r) = \{(b_1, b_2, 2c) \mid b_1 + 2 = b_2\}$, by the shift coupling (Def. 3.6). ■

$$\frac{}{\Delta \wedge \omega = 0 \rightarrow inv_{\ell_{en}}(\boldsymbol{v}, 0) \in C}\ \text{INIT} \qquad \frac{}{inv_{\ell_{ret}}(\boldsymbol{v}, \omega) \rightarrow \omega \leqslant \varepsilon \wedge (v_{r_1} = \iota \Rightarrow v_{r_2} = \iota) \in C}\ \text{DPRIV}$$

$$\frac{e = (i, j) \in L \qquad \mathit{st}_e = v \leftarrow exp}{inv_i(\boldsymbol{v}, \omega) \wedge v_1' = expr_1 \wedge v_2' = expr_2 \rightarrow inv_j(\boldsymbol{v}[v'], \omega) \in C}\ \text{ASSIGN}$$

$$\frac{e = (i, j) \in L \qquad \mathit{st}_e = \text{assume}(bexp)}{inv_i(\boldsymbol{v}, \omega) \wedge bexpr_1 \wedge bexpr_2 \rightarrow inv_j(\boldsymbol{v}, \omega) \in C}\ \text{ASSUME} \qquad \frac{e = (i, j) \in L \qquad \mathit{st}_e = \text{assume}(bexp)}{inv_i(\boldsymbol{v}, \omega) \rightarrow bexpr_1 \equiv bexpr_2 \in C}\ \text{ASSUME-S}$$

$$\frac{e = (i, j) \in L \qquad \mathit{st}_e = v \sim dexp}{inv_i(\boldsymbol{v}, \omega) \wedge strat_e(\boldsymbol{v}, v_1', v_2', \theta) \rightarrow inv_j(\boldsymbol{v}[v'], \omega + \theta) \in C}\ \text{STRAT} \qquad \frac{e = (i, j) \in L \qquad \mathit{st}_e = v \sim dexp}{dexp_1 \ \leftrightsquigarrow^{strat_e(\boldsymbol{v}, -, -, -)} \ dexp_2 \in C}\ \text{COUPLE}$$

We use $\boldsymbol{v}[v']$ to denote the vector $\boldsymbol{v}$ but with variables v_1 and v_2 replaced by their primed versions, v_1' and v_2'.

Fig. 4. Generating Horn Modulo Coupling Constraints

5.2 Generating Horn Clauses from Programs

For a program P, we generate clauses C specifying a winning coupling strategy $\{\tau_j\}_{j \in D_{v_r}}$; as we saw in Thm. 4.7, a winning coupling strategy implies ε-differential privacy. A winning coupling strategy ensures that the coupled postcondition $post_{\tau_j}(\Delta \times \{0\}, \sigma)$ satisfies certain conditions for every trace $\sigma \in \Sigma(P)$. We thus generate a set of constraints whose solutions capture all sets $post_{\tau_j}(\Delta \times \{0\}, \sigma)$. Since the set $\Sigma(P)$ is potentially infinite (due to loops), we generate a recursive system of constraints. Rather than find a coupling strategy separately for each possible output value j, we will parameterize our constraints by a logical variable ι representing a possible output.

Invariant and Strategy Relations. Our generated constraints C mention two unknown relations.

- $inv_i(\boldsymbol{v}, \omega)$ encodes the *coupled invariant* at location $i \in L$ in the program. This relation captures the set of coupled states $post_{\tau_j}(\Delta \times \{0\}, \sigma)$ for traces σ that begin at location ℓ_{en} and end at location i. The first-order variables $\boldsymbol{v}$ model two copies of program variables for the two executions of P, tagged with subscript 1 or 2, respectively, along with the logical variable ι representing a possible return value in D_{v_r}. The variable ω models the accumulated cost for the particular coupled states and program location.
- $strat_e(\boldsymbol{v}, v_1', v_2', \theta)$ encodes the coupling strategy for sampling statement st_e. If the values of $\boldsymbol{v}$ model two program states, $strat_e$ encodes a coupling between the distributions in st_e.

Constraint Generation. Given a program P and adjacency relation Δ, the rules in Fig. 4 define a set C of constraints. The rule COUPLE generates a coupling constraint for a sampling statement; all other rules generate standard constrained Horn clauses.

Before walking through the rules, we first set some notation. We use exp_1 (resp. exp_2) to denote exp with all its variables tagged with subscript 1 (resp. 2). As is standard, we assume there is a one-to-one correspondence between expressions in our language and our first-order theory, e.g., if $bexp$ is $x > 0$ and x is of type $\mathbb{Z}$, then we treat $bexp_1$ as the constraint $x_1 > 0$ in the theory of linear integer arithmetic. We assume that the adjacency relation Δ is a formula in our first-order theory. Finally, technically there are two distinct input variables ε_1 and ε_2 representing the target privacy

level. Since these variables are assumed to be equal in any two adjacent states, we will simply use a single variable ε in the constraints.

Now, we take a closer look at the constraint generation rules. The first two rules describe the initial and final states: the rule INIT specifies that the invariant at ℓ_{en} contains all adjacent states and ω is 0. The rule DPRIV states that the invariant at ℓ_{ret} satisfies the ε-DP conditions in Thm. 4.7 for every return value ι and every $\varepsilon > 0$.

The next three rules describe the coupled postcondition for deterministic statements. The rule ASSIGN encodes the effect of executing $v \leftarrow exp$ in the two executions of the program. Primed variables, e.g., v_1', to denote the modified (new) value of v_1 after assignment. The rule ASSUME encodes effects of assume statements, and ASSUME-S ensures both processes are synchronized at assume statements.

The last two rules encode sampling statements. The rule STRAT generates a clause that uses the coupling encoded by $strat_e$ to constrain the values of v_1' and v_2' and increments the privacy cost ω by θ. The rule COUPLE generates a coupling constraint specifying that $strat_e$ encodes a coupling of the distributions in the two executions. We interpret the distribution expressions $dexp_1, dexp_2$ as distribution families, parameterized by the state.

Example 5.5. For illustration, let us walk through the constraints generated for the simple program from Ex. 4.4, reproduced below. Assume the adjacency relation is $|x_1 - x_2| \leqslant 1 \wedge \varepsilon > 0$, and let the vector $\boldsymbol{v}$ contain the variables $\{x_1, x_2, \varepsilon, \iota\}$. We write $\boldsymbol{v}[x']$ for $\boldsymbol{v}[x_1 \mapsto x_1', x_2 \mapsto x_2']$ (as described at the bottom of Fig. 4). Then, the following constraints are generated by the indicated rules from Fig. 4.

1: $x \leftarrow x + 10$
2: $x \sim \mathsf{Lap}_\varepsilon(x)$
3: **return** x

$$|x_1 - x_2| \leqslant 1 \wedge \varepsilon > 0 \wedge \omega = 0 \longrightarrow inv_1(\boldsymbol{v}, \omega) \quad \text{INIT}$$

$$inv_1(\boldsymbol{v}, \omega) \wedge x_1' = x_1 + 10 \wedge x_2' = x_2 + 10 \longrightarrow inv_2(\boldsymbol{v}[x'], \omega) \quad \text{ASSIGN}$$

$$inv_2(\boldsymbol{v}, \omega) \wedge strat(\boldsymbol{v}[x'], x_1', x_2', \theta) \longrightarrow inv_3(\boldsymbol{v}[x'], \omega + \theta) \quad \text{STRAT}$$

$$inv_3(\boldsymbol{v}, \omega) \longrightarrow \omega \leqslant \varepsilon \wedge (x_1 = \iota \Rightarrow x_2 = \iota) \quad \text{DPRIV}$$

$$\mathsf{Lap}_\varepsilon(x_1) \leftrightsquigarrow^{strat(\boldsymbol{v}, -, -, -)} \mathsf{Lap}_\varepsilon(x_2) \quad \text{COUPLE}$$

The relation *strat* describes the coupling for the single sampling statement. ■

Soundness. To connect our constraint system to winning coupling strategies, the main soundness lemma states that a satisfying assignment ρ of C encodes a winning coupling strategy.

LEMMA 5.6. *Let $\rho \models C$, where C is generated for a program P and adjacency relation Δ. Define a family of coupling strategies $\{\tau_j\}_{j \in D_{v_r}}$ for P by*

$$\tau_j(\mathfrak{s}_e, s_1, s_2) = \{(a, a', c) \mid (\boldsymbol{q}_j, a, a', c) \in \rho(strat_e)\}$$

for every sampling statement $\mathfrak{s}_e$ in P and every pair of states (s_1, s_2), where $\boldsymbol{q}_j$ replaces each tagged program variable in $\boldsymbol{v}$ with the value in s_1 and s_2 respectively, and sets the output variable ι to j. Then, $\{\tau_j\}_j$ is a winning coupling strategy.

Now soundness is immediate by Lem. 5.6 and Thm. 4.7.

THEOREM 5.7 (SOUNDNESS OF CONSTRAINTS). *Let C be a system of constraints generated for program P and adjacency relation Δ, as per Fig. 4. If C is satisfiable, then P is ε-differentially private.*

6 PARAMETERIZED COUPLING FAMILIES

In this section, we demonstrate how to transform the probabilistic coupling constraints into more standard logical constraints. We introduce parameterized families of couplings (§ 6.1) and then show how to use them to eliminate coupling constraints (§ 6.2).

Table 1. Coupling families (selection)

Name	Distribution family	Precondition	Coupling family	Coupling params.
Null (Lap)	$\mathsf{Lap}_\varepsilon[z_1], \mathsf{Lap}_\varepsilon[z_2]$	–	$\Lambda_\varnothing[z_1, z_2]$	–
Shift (Lap)	$\mathsf{Lap}_\varepsilon[z_1], \mathsf{Lap}_\varepsilon[z_2]$	–	$\Lambda_{+k}[z_1, z_2]$	$k \in \mathbb{Z}$
Choice (Lap)	$\mathsf{Lap}_\varepsilon[z_1], \mathsf{Lap}_\varepsilon[z_2]$	$NO(\Pi, \Lambda_{+k}[z_1, z_2], \Lambda_\varnothing[z_1, z_2])$	$(\Pi \ ? \ \Lambda_{+k}[z_1, z_2] : \Lambda_\varnothing[z_1, z_2])$	$k \in \mathbb{Z}, \Pi \subseteq \mathbb{Z}$
Null (Exp)	$\mathsf{Exp}_\varepsilon[z_1], \mathsf{Exp}_\varepsilon[z_2]$	–	$\Lambda_\varnothing[z_1, z_2]$	–
Shift (Exp)	$\mathsf{Exp}_\varepsilon[z_1], \mathsf{Exp}_\varepsilon[z_2]$	$z_2 - z_1 \leqslant k$	$\Lambda'_{+k}[z_1, z_2]$	$k \in \mathbb{Z}$

6.1 Coupling Families

Since coupling strategies select couplings for distribution expressions instead of purely mathematical distributions, it will be useful to consider couplings between two families of distributions M_1, M_2 parametrized by variables, rather than just between two concrete distributions.

Definition 6.1 (Coupling families). Let M_1, M_2 be two distribution families with distribution parameters in some set W. Then, a *coupling family* $\Lambda[w_1, w_2]$ for M_1 and M_2 assigns a coupling

$$M_1[w_1] \leftrightsquigarrow^{\Lambda[w_1, w_2]} M_2[w_2]$$

for all $(w_1, w_2) \in \mathit{pre}_\Lambda$, where $\mathit{pre}_\Lambda \subseteq W \times W$ is a *precondition* on the parameter combinations.

We use square brackets to emphasize the distribution parameters. ■

Besides distribution parameters—which model the state of the distributions—couplings can also depend on logical parameters independent of the state. We call this second class of parameters *coupling parameters*. Table 1 lists the selection of couplings we discuss in this section along with their preconditions and coupling parameters; the distribution parameters are indicated by the square brackets.

Null and Shift Coupling Families. In § 3, we saw two examples of couplings for the Laplace and exponential distributions: the shift coupling (Def. 3.6) and the null coupling (Def. 3.7). These couplings are examples of coupling families in that they specify a coupling for a family of distributions. For instance, the Laplace distribution has the mean value z as a distribution parameter; we will write $\mathsf{Lap}_\varepsilon[z]$ instead of $\mathsf{Lap}_\varepsilon(z)$ when we want to emphasize this dependence.[4] The null and shift coupling families couple the distribution families $\mathsf{Lap}_\varepsilon[z_1], \mathsf{Lap}_\varepsilon[z_2]$ for any two $z_1, z_2 \in \mathbb{Z}$. For example, for a fixed z_1, z_2, we have the shift coupling $\mathsf{Lap}_\varepsilon(z_1) \leftrightsquigarrow^{\Lambda_{+k}} \mathsf{Lap}_\varepsilon(z_1)$, where $\Lambda_{+k} = \{(n_1, n_2, |k + z_1 - z_2|\varepsilon) \mid n_1 + k = n_2\}$. Notice that $k \in \mathbb{Z}$ is a coupling parameter; depending on what we set it to, we get different couplings.

In the case of the exponential distribution, the shift coupling family couples the families $\mathsf{Exp}_\varepsilon[z_1]$ and $\mathsf{Exp}_\varepsilon[z_2]$ under the precondition that $z_2 - z_1 \leqslant k$.

Choice Coupling. We will also use a novel coupling construction, the *choice coupling*, which allow us to combine two couplings by using a predicate on the first sample space to decide which coupling to apply. If the two couplings satisfy a *non-overlapping condition*, the result is again a coupling. This construction is inspired by ideas from Zhang and Kifer [2017], who demonstrate how the pairing between samples (*randomness alignment*) can be selected depending on the result of the first sample. These richer couplings can simplify privacy proofs (and hence solutions to invariants in our constraints), in some cases letting us construct a single proof that works for all possible output values instead of building a different proof for each output value.

[4] Properly speaking ε is also a parameter; to reduce notation, we will suppress this dependence and treat ε as a constant for this section.

Lemma 6.2 (Choice coupling). *Let $\mu_1 \in dist_{\downarrow}(B)$, $\mu_2 \in dist_{\downarrow}(B)$ be two sub-distributions, and let Π be a predicate on B. Suppose that the couplings $\mu_1 \leftrightsquigarrow^{\Lambda} \mu_2$ and $\mu_1 \leftrightsquigarrow^{\Lambda'} \mu_2$ satisfy the following* non-overlapping *condition: for every $a_1 \in \Pi, a_1' \notin \Pi$, and $a_2 \in B$, $(a_1, a_2, -) \in \Lambda$ and $(a_1', a_2, -) \in \Lambda'$ do not both hold. In other words, there is no element a_2 that is related under Λ to an element in Π and related under Λ' to an element not in Π. We abbreviate the non-overlapping condition as $NO(\Pi, \Lambda, \Lambda')$. Then the following relation*

$$(\Pi \ ? \ \Lambda : \Lambda') \triangleq \{(a_1, a_2, c) \mid (a_1 \in \Pi \implies \Lambda(a_1, a_2, c)) \wedge (a_1 \notin \Pi \implies \Lambda'(a_1, a_2, c))\}$$

is a coupling $\mu_1 \leftrightsquigarrow^{(\Pi ? \Lambda : \Lambda')} \mu_2$.

Proof. Let (μ_L, μ_R) be witnesses for the coupling Λ, and let (μ_L', μ_R') be witnesses for the coupling Λ'. Then we can define witnesses ν_L, ν_R for the choice coupling $(\Pi \ ? \ \Lambda : \Lambda')$:

$$\nu_L(a_1, a_2) \triangleq \begin{cases} \mu_L(a_1, a_2) & : a_1 \in \Pi \\ \mu_L'(a_1, a_2) & : a_1 \notin \Pi \end{cases} \quad \text{and} \quad \nu_R(a_1, a_2) \triangleq \begin{cases} \mu_R(a_1, a_2) & : a_1 \in \Pi \\ \mu_R'(a_1, a_2) & : a_1 \notin \Pi. \end{cases}$$

The support and distance conditions follow from the support and distance conditions for (μ_L, μ_R) and (μ_L', μ_R'). The first marginal condition for ν_L follows from the first marginal conditions for μ_L and μ_L', while the second marginal condition for ν_R follows from the second marginal conditions for μ_R and μ_R' combined with the non-overlapping condition on Π, Λ, and Λ'. ∎

We can generalize a choice coupling to a coupling family. Suppose we have two coupling families $\Lambda[.], \Lambda'[.]$ for some distribution families $M_1[.]$, $M_2[.]$, then we can generate a coupling family $(\Pi, \Lambda[.], \Lambda'[.])$ whose coupling parameters are the predicate Π and the parameters of the two families. (Note that Π could also have parameters.) In the example below, and in Table 1, we give one possible choice coupling for the Laplace distribution. An analogous construction applies for the exponential distribution

Example 6.3. Consider the following two coupling families for the Laplace distribution family: the shift coupling $\Lambda_{+k}[z_1, z_2]$ and the null coupling $\Lambda_{\varnothing}[z_1, z_2]$. Then, we have the coupling family

$$\mathsf{Lap}_{\varepsilon}[z_1] \leftrightsquigarrow^{(\Pi ? \Lambda_{+k}[z_1, z_2] : \Lambda_{\varnothing}[z_1, z_2])} \mathsf{Lap}_{\varepsilon}[z_2].$$

under the precondition $NO(\Pi, \Lambda_{+k}[z_1, z_2], \Lambda_{\varnothing}[z_1, z_2])$. The parameters of this coupling family are the predicate $\Pi \subseteq \mathbb{Z}$ and the parameter $k \in \mathbb{Z}$ of $\Lambda_{+k}[z_1, z_2]$. For one possible instantiation, let Π be the predicate $\{x \in \mathbb{Z} \mid x \geqslant 0\}$ and let $k = 1$. Then if $|z_1 - z_2| \leqslant 1$, the non-overlapping condition holds and $(\Pi \ ? \ \Lambda_{+1}[z_1, z_2] : \Lambda_{\varnothing}[z_1, z_2])$ is a coupling. ∎

6.2 Coupling Families as Templates

Now, we can transform HMC constraints and eliminate coupling constraints by *restricting* solutions of coupling constraints to use the previous coupling families. We model coupling parameters and the coupling strategy by Horn clauses with *uninterpreted functions*.

Horn Clauses with Uninterpreted Functions. Recall our Horn clauses may mention uninterpreted relation symbols R. We now assume an additional set F of *uninterpreted function* symbols, which can appear in interpreted formulas φ in the body/head of a clause C. A function symbol $f \in F$ of *arity* n can be applied to a vector of variables of length n; for example, consider the clause

$$r_1(x) \wedge f(x, y) = z \longrightarrow r_2(z)$$

In addition to mapping each relation symbol $r \in R$ to a relation, an interpretation ρ now also maps each function symbol $f \in F$ to a function definable in the theory—intuitively, an expression. We say that $\rho \models C$ iff ρC is valid for all $C \in \mathcal{C}$, where ρC also replaces every function application $f(\boldsymbol{v})$ by $\rho(f(\boldsymbol{v}))$.

$$\begin{aligned}
\textsc{enc}(\Lambda_{\varnothing}[z_1, z_2]) &\triangleq d_1 - z_1 = d_2 - z_2 \wedge \theta = 0 \\
\textsc{enc}(\Lambda_{+k}[z_1, z_2]) &\triangleq d_1 + f_k() = d_2 \wedge \theta = |z_1 - z_2 + f_k()| \cdot \varepsilon \\
\textsc{enc}(\Pi \ ? \ \Lambda_{+k}[z_1, z_2] : \Lambda_{\varnothing}[z_1, z_2]) &\triangleq (f_b(d_1) \Rightarrow \textsc{enc}(\Lambda_{+k}[z_1, z_2])) \wedge (\neg f_b(d_1) \Rightarrow \textsc{enc}(\Lambda_{\varnothing}[z_1, z_2])) \\
\textsc{enc}(\Lambda'_{+k}[z_1, z_2]) &\triangleq d_1 + f_k() = d_2 \wedge \theta = (z_1 - z_2 + f_k()) \cdot \varepsilon \\
\textsc{enc}(pre_{\Lambda'_{+k}[z_1, z_2]}) &\triangleq z_2 - z_1 \leqslant f_k() \\
\textsc{enc}(pre_{(\Pi ? \Lambda_{+k}[z_1, z_2] : \Lambda_{\varnothing}[z_1, z_2])}) &\triangleq (f_b(d_1) \wedge \neg f_b(d'_1) \wedge \textsc{enc}(\Lambda_{+k}[z_1, z_2]) \wedge \textsc{enc}(\Lambda_{\varnothing}[z_1, z_2])') \Rightarrow d_2 \neq d'_2
\end{aligned}$$

Fig. 5. Encodings of coupling families. We use φ' to denote φ with every variable x replaced by x'.

Example 6.4. To give an intuition for the semantics, consider the two simple clauses:

$$C_1 : x = y - 5 \longrightarrow r(x, y) \qquad\qquad C_2 : r(x, y) \wedge f(x) = z \longrightarrow z > y$$

A possible satisfying assignment ρ maps $r(x, y)$ to the formula $x = y - 5$ (ensuring that the first clause is valid) and $f(x)$ to the expression $x + 6$ (ensuring that the second clause is valid). ■

Roughly, we can view the problem of solving a set of Horn clauses $\{C_1, \ldots, C_n\}$ as solving a formula of the form $\exists f_1, \ldots, f_m. \exists r_1, \ldots, r_l. \bigwedge_i C_i$: Find an interpretation of f_j and r_k such that $\bigwedge_i C_i$ is valid. In our setting, f_j will correspond to *pieces* of the coupling strategy and r_k will be the invariants showing that the coupling strategy establishes differential privacy.

Transforming Coupling Constraints. To transform coupling constraints into Horn clauses with uninterpreted functions, we encode each coupling family $\Lambda[w_1, w_2]$ as a first-order formula, $\textsc{enc}(\Lambda[w_1, w_2])$, along with its precondition. The encodings for the families we consider from Table 1 are shown in Fig. 5, where $f_k()$ is a fresh nullary (constant) uninterpreted function that is shared between the encoding of a family and its precondition. Similarly, $f_b(\cdot)$ is a fresh uninterpreted function ranging over booleans, representing a predicate.

With the encodings of coupling families in place, we are ready to define the transformation.

Definition 6.5 (Coupling constraint transformer). Suppose we have a coupling constraint

$$M_1(\boldsymbol{v}_1) \leftrightsquigarrow^{r(\boldsymbol{v}, -, -, -)} M_2(\boldsymbol{v}_2).$$

Suppose we have a set of possible coupling families $\Lambda^1[\boldsymbol{v}_1, \boldsymbol{v}_2], \ldots, \Lambda^n[\boldsymbol{v}_1, \boldsymbol{v}_2]$ for $M_1[\boldsymbol{v}_1], M_2[\boldsymbol{v}_2]$. We transform the coupling constraint into the following set of Horn clauses, where we use $A \leftrightarrow B$ to denote the pair of Horn clauses $A \longrightarrow B$ and $B \longrightarrow A$, and we use f to denote a fresh uninterpreted function with range $\{1, \ldots, n\}$:

$$\left(\bigwedge_{i=1}^{n} f(\boldsymbol{v}) = i \implies \textsc{enc}(\Lambda^i[\boldsymbol{v}_1, \boldsymbol{v}_2])\right) \longleftrightarrow r(\boldsymbol{v}, d_1, d_2, \theta)$$

$$f(\boldsymbol{v}) = 1 \longrightarrow \textsc{enc}(pre_{\Lambda^1}[\boldsymbol{v}_1, \boldsymbol{v}_2]) \qquad \cdots \qquad f(\boldsymbol{v}) = n \longrightarrow \textsc{enc}(pre_{\Lambda^n}[\boldsymbol{v}_1, \boldsymbol{v}_2])$$

Without loss of generality, we assume that variables d_1, d_2, θ do not appear in $\boldsymbol{v}, \boldsymbol{v}_1, \boldsymbol{v}_2$.

Intuitively, the first clause selects which of the n couplings we pick, depending on the values $\boldsymbol{v}$. The remaining clauses ensure that whenever we pick a coupling Λ^i, we satisfy its precondition. ■

Example 6.6. Recall our Report Noisy Max example from § 2. There, we encoded the single sampling statement using the following coupling constraint:

$$C_4 : \mathsf{Lap}_{\varepsilon/2}(q_1[i_1]) \leftrightsquigarrow^{strat(\boldsymbol{v}, -, -, -)} \mathsf{Lap}_{\varepsilon/2}(q_2[i_2])$$

Suppose we take the null and shift coupling families for Laplace as our possible couplings. Our transformation produces:

$$\bigwedge \begin{array}{l} f(\boldsymbol{v}) = 1 \implies \left(d_1' - d_2' = q_1[i_1] - q_2[i_2] \wedge \theta = 0\right) \\ f(\boldsymbol{v}) = 2 \implies \left(d_1' + f_k() = d_2' \wedge \theta = |q_1[i_1] - q_2[i_2] + f_k()| \cdot (\varepsilon/2)\right) \end{array} \longleftrightarrow strat(\boldsymbol{v}, d_1', d_2', \theta)$$

We omit clauses enforcing the trivial precondition (*true*) for these coupling families.

The first conjunct on the left encodes application of the null coupling; the second conjunct encodes the shift coupling. The unknowns are the function f and the constant function f_k. As we sketched in § 2, our tool discovers the interpretation $ite(i_1 \neq \iota, 1, 2)$ for $f(\boldsymbol{v})$ and the constant 1 for $f_k()$, where $ite(b, a, a')$ denotes the expression that is a if b is true and a' otherwise. ■

The following theorem formalizes soundness of our transformation.

Theorem 6.7 (Soundness of transformation). *Let C be a set of Horn clauses with coupling constraints. Let C' be the set C with coupling constraints replaced by Horn clauses, as in Def. 6.5. Then, if C' is satisfiable, C is satisfiable.*

A Note on Coupling Families. Our transformation uses a finite set of coupling families as templates for solutions of couplings constraints. This, of course, restricts the space of solutions: C may be satisfiable, while C' may not be. However, the coupling families we propose are rather general: they capture all couplings proposed in the ε-DP verification literature. Our approach is just as expressive as full-blown program logics, like apRHL [Barthe et al. 2016b].

7 IMPLEMENTATION AND EVALUATION

We now describe the algorithms we implement for solving Horn clauses, detail our implementation, and evaluate our technique on well-known algorithms from the differential privacy literature.

7.1 Solving Horn Clauses with Uninterpreted Functions

Solving Horn clauses with uninterpreted functions is equivalent to a program synthesis/verification problem, where we have an *incomplete* program with holes and we want to fill the holes to make the program satisfy some specification. In our implementation we employ a *counterexample-guided inductive synthesis* (CEGIS) [Solar-Lezama et al. 2006] technique to propose a solution for the uninterpreted functions F, and a fixed-point computation with *predicate abstraction* [Graf and Saïdi 1997] to verify if the Horn clauses are satisfiable for the candidate interpretation of F. Since these techniques are relatively standard in the program synthesis literature, we describe our solver at a high level in this section.

```
1: fun SOLVE(C)
2:     n ← 1
3:     while true do
4:         ρ^F ← SYNTH(∃F. ⋀ C^n)
5:         res ← VERIFY(ρ^F C)
6:         if res is SAT then
7:             return SAT
8:         n ← n + 1
```

Fig. 6. Algorithm for solving Horn clauses with uninterpreted functions

Synthesize–Verify Loop. Algorithm SOLVE (Fig. 6) employs a synthesize–verify loop as follows:

Coupling Strategy Synthesis. In every iteration of the algorithm, the set of clauses C are *unrolled* into a set of clauses C^n, a standard technique in Horn-clause solving roughly corresponding to unrolling program loops up to a finite bound—see, for example, the algorithm of McMillan and Rybalchenko [2013]. We assume that there is a single clause $Q \in C$ whose head is not a relation application—we call it the *query* clause. The unrolling starts with clause Q, and creates a fresh copy of every clause whose head is a relation that appears in the body of Q, and so on, recursively, up to depth n. For an example, consider the following three clauses: $r(x) \longrightarrow x > 0$ and $r(x) \wedge f(x) = x' \longrightarrow r(x')$. If we unroll these clauses up to $n = 3$, we get the following

clauses: (*i*) $r^3(x) \longrightarrow x > 0$, (*ii*) $r^2(x) \wedge f(x) = x' \longrightarrow r^3(x')$, and (*iii*) $r^1(x) \wedge f(x) = x' \longrightarrow r^2(x')$, where r^i are fresh relation symbols (copies of r).
Once we have non-recursive clauses C^n, we can construct a formula of the form

$$\exists f_1, \ldots, f_n. \exists r_1, \ldots, r_m. \bigwedge_{C \in C^n} C$$

where f_i and r_i are the function and relation symbols appearing in the unrolled clauses. Since C^n are non-recursive, we can rewrite them to remove the relation symbols. This is a standard encoding in Horn clause solving that is analogous to encoding a loop-free program as a formula—as in VC generation and bounded model checking—and we do not detail it here. We thus transform $\bigwedge_{C \in C^n} C$ into a constraint of the form $\exists f_1, \ldots, f_n. \forall \boldsymbol{x}. \varphi$ which can be solved using program synthesis algorithms (recall that variables are implicitly universally quantified). Specifically, for SYNTH, we employ a version of the symbolic synthesis algorithm due to Gulwani et al. [2011]. The synthesis phase generates an interpretation ρ^F of the function symbols, a winning coupling strategy that proves ε-DP for adjacent inputs that terminate in n iterations.

Coupling Strategy Verification. This second phase verifies whether the synthesized strategy is indeed a winning coupling strategy for all pairs of adjacent inputs by checking if $\rho^F C$ is satisfiable. That is, we plug in the synthesized values of F into C, resulting in a set of Horn clauses with no uninterpreted functions—only invariant relation symbols inv_i—which can be solved with a fixed-point computation. The result of VERIFY could be SAT, UNSAT, or UNKNOWN, due to the undecidability of the problem. If the result is SAT, then we know that ρ^F encodes a winning coupling strategy and we are done. Otherwise, we try unrolling further.
While there are numerous Horn-clause solvers available, we have found that they are unable to handle the clauses we encountered. We thus implemented a custom solver that uses *predicate abstraction*. We detail our implementation and rationalize this decision below.

7.2 Implementation Details

We have implemented our approach by extending the FairSquare probabilistic verifier [Albarghouthi et al. 2017]. Our implementation takes a program P in a simple probabilistic language with integers and arrays over integers. An adjacency relation is provided as a first-order formula over input program variables. The program P is then translated into a set of Horn clauses over the combined theories of linear real/integer arithmetic, arrays, and uninterpreted functions.

Instantiations of the Synthesizer (SYNTH). To find an interpretation of functions F, our synthesis algorithm requires a grammar of expressions to search through. Constant functions can take any integer value. In the case of Boolean and n-valued functions, we instantiate the synthesizer with a grammar over Boolean operations, e.g., $\bullet_1 > \bullet_2$ or $\bullet_1 \neq \bullet_2$, where $\bullet_1$ and $\bullet_2$ can be replaced by variables or numerical expressions over variables. In all of our benchmarks, we find that searching for expressions with ASTs of depth 2 is sufficient to finding a winning coupling strategy.

Instantiations of Predicate Abstraction (VERIFY). To compute invariants inv_i, we employ predicate abstraction [Graf and Saïdi 1997]. In our initial experiments, we attempted to delegate this process to existing Horn clause solvers. Unfortunately, they either diverged or could not handle the theories we use.[5] We thus built a Horn clause solver on top of the Z3 SMT solver and instantiated it with a large set of predicates *Preds* over a rich set of templates over pairs of variables. The templates

[5]Duality [McMillan and Rybalchenko 2013], which uses *tree interpolants*, diverged even on the simplest example. This is due to the interpolation engine not being optimized for relational verification; for instance, in all examples we require invariants containing multiple predicates of the form $v_1 = v_2$ or $|v_1 - v_2| \leqslant 1$, where v_1 and v_2 are copies of the same variable. SeaHorn [Gurfinkel et al. 2015], HSF [Grebenshchikov et al. 2012a], and Eldarica [Hojjat et al. 2012] have limited or no support for arrays, and could not handle our benchmarks.

```
fun SmartSum(q)
    next, n, i, sum ← 0
    r ← []
    while i < |q| do
        sum ← sum + q[i]
        if (i + 1) % M = 0 then
            n ~ Lap_ε(n + sum)
            sum, next ← 0, n
        else
            next ~ Lap_ε(next + q[i])
        r ← next :: r
        i ← i + 1
    return r
```

```
fun ExpMech(d, qscore)
    i ← 1
    bq ← 0
    while i ⩽ R do
        s ← qscore(i, d)
        cq ← Exp_ε/2(s)
        if cq > bq ∨ i = 1 then
            max ← i
            bq ← cq
        i ← i + 1
    return max
```

```
fun NumericSparseN(qs, T, N)
    r ← []
    i, ct ← 0
    t ~ Lap_ε/3(T)
    while i < |qs| ∧ ct < N do
        n ~ Lap_ε/6N(qs[i])
        if n > t then
            ans ~ Lap_ε/3N(qs[i])
            r ← (i, ans) :: r
            ct ← ct + 1
        i ← i + 1
    return r
```

Fig. 7. Three representative benchmarks (in practice, lists are encoded as arrays)

are of the form: $\bullet_1 > \bullet_2$, $\bullet_1 = \bullet_2$, $\bullet_1 + k = \bullet_2$, $|\bullet_1 - \bullet_2| \leqslant k$. By instantiating $\bullet$ with program variables and k with constants, we generate a large set of predicates. We also use the predicates appearing in `assume` statements. To track the upper bound on ω, we use a class of predicates of the form $\omega \leqslant \sum_{j=1}^{n} \bullet_j \cdot iexp_j$ that model every increment to the privacy cost, where $iexp_j$ is the scale parameter in the jth sampling statement in the program ($\mathsf{Lap}_{iexp_j}(-)$ or $\mathsf{Exp}_{iexp_j}(-)$), and $\bullet_j$ is instantiated as an expression of the cost of the chosen couplings for the jth statement, potentially multiplied by a positive program variable if it occurs in a loop. For instance, if we have the shift coupling, we instantiate $\bullet_j$ with $|k + z_1 - z_2|$; if we use the null coupling, $\bullet_j$ is 0. We use all possible combinations of $\bullet_j$ and eliminate any non-linear predicates (see more below).

Since our invariants typically require disjunctions we employ boolean predicate abstraction, an expensive procedure requiring exponentially many SMT calls in the size of the predicates in the worst case. To do so efficiently, we use the AllSMT algorithm of Lahiri et al. [2006], as implemented in the MathSAT5 SMT solver [Cimatti et al. 2013].

Handling Non-linearity. Observe that in the definition of shift couplings (Def. 3.6), the parameter ε is multiplied by an expression, resulting in a non-linear constraint (a theory that is not well-supported by SMT solvers, due to its complexity). To eliminate non-linear constraints, we make the key observation that it suffices to set ε to 1: Since all constraints involving ε are summations of coefficients of ε, we only need to track the coefficients of ε.

Handling Conditionals. Winning coupling strategies (Thm. 4.7) assume synchronization at each conditional statement. In our implementation, we only impose synchronization at loop heads by encoding conditional code blocks (with no sampling statements) as monolithic instructions, as in *large-block encoding* [Beyer et al. 2009]. This enables handling examples like Report Noisy Max (§ 2), where the two processes may not enter the same branch of the conditional under the strategy.

7.3 Differentially Private Algorithms.

For evaluation, we automatically synthesized privacy proofs for a range of algorithms from the differential privacy literature. The examples require a variety of different kinds of privacy proofs. Simpler examples follow by composition, while more complex examples require more sophisticated arguments. Fig. 7 shows the code for three of the more interesting algorithms, which we discuss below; Table 2 presents the full collection of examples.

Two-Level Counter (SmartSum). The SmartSum algorithm [Chan et al. 2011; Dwork et al. 2010] was designed to continually publish aggregate statistics while maintaining privacy—for example, continually releasing the total number of visitors to a website. Suppose that we have a list of inputs q, where $q[i]$ denotes the total number of visitors in hour i. SmartSum releases a list of all running

sums: $q[0]$, $q[0] + q[1]$, $\ldots$, $\sum_{j=0}^{i} q[j]$, $\ldots$. Rather than adding separate noise to each $q[i]$ or adding separate noise to each running sum—which would provide weak privacy guarantees or inaccurate results—SmartSum chunks the inputs into blocks of size M and adds noise to the sum of each block. Then, each noisy running sum can be computed by summing several noised blocks and additional noise for the remaining inputs. By choosing the block size carefully, this approach releases all running sums with less noise than more naïve approaches, while ensuring ε-DP.

Discrete Exponential Mechanism (ExpMech). The exponential mechanism [McSherry and Talwar 2007] is used when (*i*) the output of an algorithm is non-numeric or (*ii*) different outputs have different *utility*, and adding numeric noise to the output would produce an unusable answer, e.g., in the case of an auction where we want to protect privacy of bidders [Dwork and Roth 2014]. ExpMech takes a database and a *quality score* (utility function) as input, where the quality score maps the database and each element of the range to a numeric score. Then, the algorithm releases the element with approximately the highest score. We verified a version of this algorithm that adds noise drawn from the exponential mechanism to each quality score, and then releases the element with the highest noisy score. This implementation is also called the *one-sided Noisy Arg-Max* [Dwork and Roth 2014].

Sparse Vector Mechanism (NumericSparseN). The Sparse Vector mechanism (also called NumericSparse in the textbook by Dwork and Roth [2014]) is used in scenarios where we would like to answer a large number of numeric queries while only paying for queries with large answers. To achieve ε-DP, NumericSparseN releases the noised values of the first N queries that are above some known threshold T, while only reporting that other queries are below the threshold, without disclosing their values.

Specifically, NumericSparseN takes a list of numeric queries qs and numeric threshold T. Each query is assumed to be 1-sensitive—its answer may differ by at most 1 on adjacent databases. The threshold T is assumed to be public knowledge, so we can model it as being equal in adjacent inputs. The Sparse Vector mechanism adds noise to the threshold and then computes a noisy answer for each query. When the algorithm finds a query where the noisy answer is larger than the noisy threshold, it records the index of the query. It also adds fresh noise to the query answer, and records the (freshly) noised answer as well. When the algorithm records N queries or runs out of queries, it returns the final list of indices and answers.

The privacy proof is interesting for two reasons. First, applying the composition theorem of privacy does prove privacy, but with an overly conservative level of ε that depends on the number of queries. A more careful proof by coupling shows that the privacy level depends only on the number of above threshold queries, potentially a large saving if most of the queries are below threshold. Second, adding *fresh* noise when estimating the answer of above threshold queries is critical for privacy—it is not private to reuse the noisy answer of the query from checking against the noisy threshold. Previous versions of this algorithm suffered from this flaw [Lyu et al. 2017].

7.4 Experimental Results

Table 3 summarizes the results of applying our implementation to the differentially private algorithms described in Table 2. For each algorithm, we established ε-DP; in case of SmartSum, we show that it is 2ε-DP, as established by Chan et al. [2011].[6] In all our examples, we transform coupling constraints (using Def. 6.5) by instantiating $strat_e$ (for every sampling sampling statement $\mathbf{s}_e$) with two choices, the null and shift coupling families of Lap or Exp—i.e., the function $f(\boldsymbol{v})$ in Def. 6.5 has range $\{1, 2\}$. While this is sufficient to find a proof for all examples, in the case of the Sparse Vector mechanism (NumericSparse*), we found that the proof requires an invariant that lies outside our

[6] Technically, this requires changing the upper bound on ω from ε to 2ε in the clause generated by rule DPRIV in Fig. 4.

Table 2. Differentially private algorithms used in our evaluation

Algorithm	Description
PartialSum	Compute the noisy sum of a list of queries.
PrefixSum	Compute the noisy sum for every prefix of a list of queries.
SmartSum	Advanced version of PrefixSum that chunks the list [Chan et al. 2011; Dwork et al. 2010].
ReportNoisyMax	Find the element with the highest quality score [Dwork and Roth 2014].
ExpMech	Variant of ReportNoisyMax using the exponential distribution [Dwork and Roth 2014; McSherry and Talwar 2007].
AboveThreshold	Find the index of the first query above threshold [Dwork and Roth 2014].
AboveThresholdN	Find the indices of the first N queries with answer above threshold [Dwork and Roth 2014; Lyu et al. 2017].
NumericSparse	Return the index and answer of the first query above threshold [Dwork and Roth 2014].
NumericSparseN	Return the indices and answers of the first N queries above threshold [Dwork and Roth 2014; Lyu et al. 2017].

decidable theory. Using the choice coupling family for those algorithms, our tool can automatically discover a much simpler invariant.

Results Overview. For each algorithm, Table 3 shows the privacy bound and the number of sampling statements (#Samples) appearing in the algorithm to give a rough idea of how many couplings must be found. The verification statistics show the number of predicates (#Preds) generated from our predicate templates, and the time spent in verify. The synthesis statistics show the size (in terms of ast nodes) of the largest formula passed to synth and number of variables appearing in it; the number of cegis iterations of the synthesis algorithm [Gulwani et al. 2011]; and the time spent in synth.

In all cases, our implementation was able to automatically establish differential privacy in a matter of minutes. However, there is significant variation in the time needed for different algorithms. Consider, for instance, NumericSparseN. While it is a more general version of AboveThresholdN, it requires less verification time. This is because the invariant required by NumericSparseN is conjunctive, allowing verify to quickly discover an invariant. The invariant required by AboveThresholdN contains multiple disjuncts, making it spend more time in boolean abstraction. (Note that an upper bound on the number of possible disjunctive invariants using n predicates is 2^{2^n}.) Synthesis time, however, is higher for NumericSparseN. This is due to the use of the choice coupling family, which has two coupling parameters and requires discovering a non-overlapping predicate P.

Detailed Discussion. For a more detailed view, let us consider the ReportNoisyMax algorithm. In § 2, we described in detail the winning coupling strategy our tool discovers. To verify that the coupling strategy is indeed winning, the example requires the following coupled invariant:

$$\mathcal{I} \triangleq (i_1 = i_2) \wedge (i_1 \leqslant \iota \implies \mathcal{I}_{\leqslant} \wedge \omega \leqslant 0) \wedge (i_1 > \iota \implies \mathcal{I}_{>} \wedge \omega \leqslant \varepsilon),$$

where

$$\mathcal{I}_{\leqslant} \triangleq ((r_1 = r_2 = \iota = 0) \vee (r_1 < \iota \wedge r_2 < \iota)) \wedge |best_1 - best_2| \leqslant 1$$
$$\mathcal{I}_{>} \triangleq r_1 = \iota \implies (r_2 = \iota \wedge best_1 + 1 = best_2).$$

At a high-level, the invariant considers two cases: (*i*) when the loop counter i_1 is less than or equal to the output ι, and (*ii*) when the loop counter i_1 has moved past the output ι. In the latter case, the invariant ensures that if $r_1 = \iota$, then $r_2 = \iota$, and, crucially, $best_1 + 1 = best_2$. Notice also how the invariant establishes that the privacy cost is upper-bounded by ε. Our tool discovers a similar invariant for ExpMech, with the strategy also establishing the precondition of the shift coupling.

Table 3. Experimental results (OS X 10.11; 4GHz Intel Core i7; 16GB RAM)

			Verification stats.		Synthesis stats.			
Algorithm	Bound	#Samples	#*Preds*	Time (s)	#vars	Formula size	#CEGIS iters.	Time (s)
PartialSum	ε	1	30	12	183	566	9	0.4
PrefixSum	ε	1	40	14	452	1246	8	1.1
SmartSum	2ε	2	44	255	764	2230	1766	579.2
ReportNoisyMax	ε	1	36	22	327	1058	35	1.5
ExpMech	ε	1	36	22	392	1200	152	5.0
AboveThreshold	ε	2	37	27	437	1245	230	7.3
AboveThresholdN	ε	3	59	580	692	1914	628	31.3
NumericSparse	ε	2	62	4	480	1446	65	3.2
NumericSparseN	ε	3	68	5	663	1958	6353	1378.9

Let us now consider a more complex example, the NumericSparseVectorN, which utilizes the choice coupling. Notice that there are 3 sampling statements in this example.

- For the first sampling statement, we use the shift coupling to ensure that $t_1 + 1 = t_2$; since $T_1 = T_2$, we incur a privacy cost of $\varepsilon/3$.
- For the second sampling statement, we use the choice coupling: When $n_1 > t_1$, we use the shift coupling to ensure that $n_1 + 1 = n_2$; otherwise, we take the null coupling. Since $qs_1[i_1]$ and $qs_2[i_2]$ may differ by one, as per the adjacency relation, we will incur a worst-case price of $\varepsilon/3$ for this statement, since we will incur $2 * \varepsilon/6N$ cost for each of the N iterations where $n_1 > t_1$.
- Finally, for the third sampling statement, use the shift coupling to ensure that $ans_1 = ans_2$, incurring a cost of $\varepsilon/3$ across the N iterations where the conditional is entered.

As such, this strategy establishes that the NumericSparseVecN is $\varepsilon = \varepsilon/3 + \varepsilon/3 + \varepsilon/3$ differentially private. It is easy to see that this is a winning coupling strategy, since whenever the first process enters the conditional and updates r, the second process also enters the conditional and updates r with the same value—as the strategy ensures that $ans_1 = ans_2$ and $i_1 = i_2$.

Summary. Our results demonstrate that our proof technique based on winning coupling strategies is (*i*) amenable to automation through Horn-clause solving, and (*ii*) is applicable to non-trivial algorithms proposed in the differential privacy literature. To the best of our knowledge, our technique is the first to automatically establish privacy for all of the algorithms in Table 3.

8 RELATED WORK

Formal Verification of Differential Privacy. Researchers have explored a broad array of techniques for static verification of differential privacy, including linear and dependent type systems [Azevedo de Amorim et al. 2014, 2017; Barthe et al. 2015; Gaboardi et al. 2013; Reed and Pierce 2010; Zhang and Kifer 2017], relational program logics [Barthe et al. 2017b, 2016a,b, 2013; Barthe and Olmedo 2013; Hsu 2017; Olmedo 2014; Sato 2016], partial evaluation [Winograd-Cort et al. 2017], and more. Barthe et al. [2016c] provide a recent survey.

Our work is inspired by the LightDP system [Zhang and Kifer 2017], which combines a relational type system for an imperative language, a novel type-inference algorithm, and a product program construction. Types in the relational type system encode randomness alignments. Numeric types have the form $\mathsf{num}_{\mathsf{d}}$, where d is a *distance expression* that can mention program variables. Roughly speaking, the distance d describes how to map samples x from the first execution to samples $x + |\mathsf{d}|$ in the second execution. Zhang and Kifer [2017] focus on randomness alignments that are

injective maps from samples to samples. Such maps also give rise to approximate couplings; for instance, if $g : \mathbb{Z} \to \mathbb{Z}$ is injective, then there exists a variable approximate coupling of two Laplace distributions $\Lambda = \{(z, g(z), c(z))\}$ for some costs $c(z)$. However, approximate couplings are more general than randomness alignments, as couplings do not require an injective map on samples.

Zhang and Kifer [2017] also show how distance expressions (and hence randomness alignments) in LightDP can be combined: they can be added and subtracted ($\mathrm{d}_1 \oplus \mathrm{d}_2$), multiplied and divided ($\mathrm{d}_1 \otimes \mathrm{d}_2$), and formed into conditional expressions ($\mathrm{d}_1 \odot \mathrm{d}_2 \ ?\ \mathrm{d}_3 : \mathrm{d}_4$, where $\odot$ is a binary comparison). This last conditional construction is highly useful for proving privacy of certain examples, and is modeled by the choice coupling in our system (§ 6). Conceptually, our work gives a better understanding of randomness alignment and approximate couplings—in some sense, randomness alignments can be seen as a particular case of (variable) approximate couplings. LightDP also uses MaxSMT to find an optimal privacy guarantee. Our approach is property-directed: we try to prove a set upper-bound. We could enrich the generated constraints with an objective function minimizing the privacy cost ω, and solve them using an optimal synthesis technique [Bornholt et al. 2016].

We also draw on ideas from Barthe et al. [2014], who, like Zhang and Kifer [2017], use a product construction for proving differential privacy. Each call to a sampling instruction is replaced by a call to a non-deterministic function, incrementing the cost in a ghost variable v_ε. However, their system cannot prove privacy beyond composition, and the cost is required to be deterministic. In particular, many of the advanced examples that we consider, like Report Noisy Max and the Sparse Vector mechanisms, are not verifiable. Our work can be seen as extending their system along three axes: (*i*) supporting richer couplings for the sampling instructions to handle more complex examples, (*ii*) reasoning about randomized privacy costs, and (*iii*) automatically constructing the proof.

Automated Verification and Synthesis. We reduced the problem of verifying ε-DP to solving Horn clauses with uninterpreted functions. Beyene et al. [2013] introduced existentially quantified Horn clauses, which were used for infinite-state CTL verification and solving ω-regular games [Beyene et al. 2014]. There, variables in the head of a clause can be existentially quantified. To solve the clauses, they *Skolemize* the existential variables and discover a solution to a Skolem relation. Using this technique, we could have adopted an alternative encoding by using existentially quantified variables to stand for unknown couplings in strategies. However, for correctness, we would need to restrict solutions for Skolem relations to be couplings—e.g., by enforcing a template for the Skolem relations.

Solving Horn clauses with uninterpreted functions is closely connected to *sketching-based* program synthesis problems [Solar-Lezama et al. 2006], where we want to find substitutions for holes in a program to ensure that the whole program satisfies a target property. One could reformulate our problem of solving Horn clauses with uninterpreted functions as solving a program synthesis problem, and, e.g., use the template-based technique of Srivastava et al. [2010] to find an inductive invariant and a coupling strategy. In our implementation, we used a synthesize-and-verify loop, a methodology that has appeared in different guises, for example, in the Sketch synthesizer [Solar-Lezama et al. 2006] and the E-HSF Horn-clause solver [Beyene et al. 2013].

Hyperproperties. There is a rich body of work on verifying relational/hyper- properties of programs, e.g., in translation-validation [Necula 2000; Pnueli et al. 1998], security problems [Clarkson and Schneider 2010; Terauchi and Aiken 2005], and Java code [Sousa and Dillig 2016]. There has been far less work on automated reasoning for probabilistic relational properties. Techniques for reasoning about quantitative information flow [Köpf and Rybalchenko 2010] typically reason about probabilities, since the property depends on a distribution on inputs. However, since there are no universally quantified variables in the property, the verification problem can be handled by *model-counting*: counting the number of executions that satisfy certain properties. The ε-DP

property quantifies over all sets of adjacent inputs and all outputs, making automated verification significantly more challenging.

9 CONCLUSIONS AND FUTURE WORK

We have presented a novel proof technique for ε-differential privacy, by finding a strategy that uses variable approximate couplings to pair two adjacent executions of a program. We formulated the set of *winning* strategies in a novel constraint system using Horn clauses and coupling constraints. By carefully restricting solutions of coupling constraints to encodings of coupling families, we can automatically prove correctness of complex algorithms from the differential privacy literature.

The next natural step would be to automate proofs of (ε, δ)-differential privacy. Approximate couplings have been used in the past to verify this more subtle version of privacy, but there are both conceptual and practical challenges in automatically finding these proofs. On the theoretical end, while we proved ε-DP by finding a coupling strategy with cost ε for each output, if we find a coupling strategy with cost (ε, δ) for each output, then δ parameters will sum up over all outputs to give the privacy level for all outputs—this can lead to a very weak guarantee, or a useless guarantee if there are infinitely many outputs. On the more practical side, in many cases the δ parameter in an (ε, δ)-private algorithm arises from an *accuracy bound* stating that a certain key sample has a small, δ probability of being too large. These bounds often have a complex form, involving division and logarithm operations. Due to these and other obstacles, to date there is no automatic system for proving (ε, δ)-privacy. Nevertheless, extending our techniques to find such proofs would be an intriguing direction for future work.

There are other possible targets of our approach beyond differential privacy. For example, we plan to adapt our technique to synthesize independence and uniformity for properties of probabilistic programs [Barthe et al. 2017a]. Additionally, we plan to combine our techniques with probabilistic resource bounds analyses to synthesize couplings that provide upper bounds on distances between probabilistic processes, e.g., to show rapid mixing of Markov chains [Barthe et al. 2017c].

ACKNOWLEDGMENTS

We thank Gilles Barthe, Marco Gaboardi, Zachary Kinkaid, Danfeng Zhang, and the anonymous reviewers for stimulating discussions and useful comments on earlier drafts of this work. This work is partially supported by the National Science Foundation CNS under Grant No. 1513694, the National Science Foundation CCF under Grant Nos. 1566015, 1704117, 1652140, the European Research Council under Grant No. 679127, and the Simons Foundation under Grant No. 360368 to Justin Hsu.

REFERENCES

Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 80 (Oct. 2017), 30 pages. https://doi.org/10.1145/3133904

Arthur Azevedo de Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. 2014. Really natural linear indexed type-checking. In *Symposium on Implementation and Application of Functional Programming Languages (IFL), Boston, Massachusetts*. ACM Press, 5:1–5:12. http://arxiv.org/abs/1503.04522

Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin-ya Katsumata, and Ikram Cherigui. 2017. A semantic account of metric preservation. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*. 545–556. http://arxiv.org/abs/1702.00374

Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2017a. Proving uniformity and independence by self-composition and coupling. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), Maun, Botswana (EPiC Series in Computing)*, Vol. 46. 385–403. https://arxiv.org/abs/1701.06477

Gilles Barthe, Thomas Espitau, Justin Hsu, Tetsuya Sato, and Pierre-Yves Strub. 2017b. ⋆-Liftings for differential privacy. In *International Colloquium on Automata, Languages and Programming (ICALP), Warsaw, Poland*. https://arxiv.org/abs/1705.00133

Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016a. Advanced probabilistic couplings for differential privacy. In *ACM SIGSAC Conference on Computer and Communications Security (CCS), Vienna, Austria*. https://arxiv.org/abs/1606.07143

Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, César Kunz, and Pierre-Yves Strub. 2014. Proving differential privacy in Hoare logic. In *IEEE Computer Security Foundations Symposium (CSF), Vienna, Austria*. 411–424. http://arxiv.org/abs/1407.2988

Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. 2015. Higher-order approximate relational refinement types for mechanism design and differential privacy. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Mumbai, India*. 55–68. http://arxiv.org/abs/1407.6845

Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016b. Proving differential privacy via probabilistic couplings. In *IEEE Symposium on Logic in Computer Science (LICS), New York, New York*. 749–758. http://arxiv.org/abs/1601.05047

Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin C. Pierce. 2016c. Programming language techniques for differential privacy. *ACM SIGLOG News* 3, 1 (Jan. 2016), 34–53. http://siglog.hosting.acm.org/wp-content/uploads/2016/01/siglog_news_7.pdf

Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2017c. Coupling proofs are probabilistic product programs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*. 161–174. http://arxiv.org/abs/1607.03455

Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella-Béguelin. 2013. Probabilistic Relational Reasoning for Differential Privacy. *ACM Transactions on Programming Languages and Systems* 35, 3 (2013), 9. http://software.imdea.org/~bkoepf/papers/toplas13.pdf

Gilles Barthe and Federico Olmedo. 2013. Beyond Differential Privacy: Composition Theorems and Relational Logic for f-divergences between Probabilistic Programs. In *International Colloquium on Automata, Languages and Programming (ICALP), Riga, Latvia (Lecture Notes in Computer Science)*, Vol. 7966. Springer-Verlag, 49–60. http://certicrypt.gforge.inria.fr/2013.ICALP.pdf

Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. 2014. A Constraint-based Approach to Solving Games on Infinite Graphs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), San Diego, California*. New York, NY, USA, 221–233.

Tewodros A Beyene, Corneliu Popeea, and Andrey Rybalchenko. 2013. Solving existentially quantified Horn clauses. In *International Conference on Computer Aided Verification (CAV), Saint Petersburg, Russia*. Springer-Verlag, 869–882.

Dirk Beyer, Alessandro Cimatti, Alberto Griggio, M. Erkan Keremoglu, and Roberto Sebastiani. 2009. Software model checking via large-block encoding. In *Formal Methods in Computer-Aided Design (FMCAD), Austin, Texas*. IEEE, 25–32.

Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. 2015. Horn clause solvers for program verification. In *Fields of Logic and Computation II*. Springer-Verlag, 24–51.

James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze. 2016. Optimizing synthesis with metasketches. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Saint Petersburg, Florida*. 775–788.

T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and continual release of statistics. *ACM Transactions on Information and System Security* 14, 3 (2011), 26. http://eprint.iacr.org/2010/076.pdf

Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. 2013. The MathSAT5 SMT Solver., Vol. 7795. Springer-Verlag, 93–107.

Michael R. Clarkson and Fred B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *IACR Theory of Cryptography Conference (TCC), New York, New York (Lecture Notes in Computer Science)*, Vol. 3876. Springer-Verlag, 265–284. https://doi.org/10.1007/11681878_14

Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential privacy under continual observation. In *ACM SIGACT Symposium on Theory of Computing (STOC), Cambridge, Massachusetts*. 715–724. http://www.mit.edu/~rothblum/papers/continalobs.pdf

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

Cynthia Dwork and Aaron Roth. 2014. *The Algorithmic Foundations of Differential Privacy*. Vol. 9. Now Publishers, Inc. 211–407 pages.

Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. 2013. Linear dependent types for differential privacy. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Rome, Italy*. 357–370. http://dl.acm.org/citation.cfm?id=2429113

Susanne Graf and Hassen Saïdi. 1997. Construction of abstract state graphs with PVS. In *International Conference on Computer Aided Verification (CAV), Haifa, Israel*. Springer-Verlag, 72–83.

Sergey Grebenshchikov, Ashutosh Gupta, Nuno P. Lopes, Corneliu Popeea, and Andrey Rybalchenko. 2012a. HSF (C): A software verifier based on Horn clauses. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Tallinn, Estonia*. Springer-Verlag, 549–551.

Sergey Grebenshchikov, Nuno P. Lopes, Corneliu Popeea, and Andrey Rybalchenko. 2012b. Synthesizing Software Verifiers from Proof Rules. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Beijing, China*. 405–416. https://doi.org/10.1145/2254064.2254112

Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. 2011. Synthesis of loop-free programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), San Jose, California*.

Arie Gurfinkel, Temesghen Kahsai, and Jorge A. Navas. 2015. SeaHorn: A Framework for Verifying C Programs (Competition Contribution). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), London, England*. 447–450.

Hossein Hojjat, Filip Konečnỳ, Florent Garnier, Radu Iosif, Viktor Kuncak, and Philipp Rümmer. 2012. A verification toolkit for numerical transition systems. In *International Symposium on Formal Methods (FM), Paris, France*. Springer-Verlag, 247–251.

Justin Hsu. 2017. *Probabilistic Couplings for Probabilistic Reasoning*. Ph.D. Dissertation. University of Pennsylvania. arXiv:cs.LO/1710.09951 https://arxiv.org/abs/1710.09951

Boris Köpf and Andrey Rybalchenko. 2010. Approximation and randomization for quantitative information-flow analysis. In *IEEE Computer Security Foundations Symposium (CSF), Edinburgh, Scotland*. 3–14.

Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. 2006. SMT techniques for fast predicate abstraction. In *International Conference on Computer Aided Verification (CAV), Seattle, Washington*. Springer-Verlag, 424–437.

Torgny Lindvall. 2002. *Lectures on the coupling method*. Courier Corporation.

Min Lyu, Dong Su, and Ninghui Li. 2017. Understanding the Sparse Vector Technique for Differential Privacy. In *International Conference on Very Large Data Bases (VLDB), Munich, Germany*, Vol. 10. 637–648. http://arxiv.org/abs/1603.01699

Kenneth L. McMillan and Andrey Rybalchenko. 2013. *Solving constrained Horn clauses using interpolation*. Technical Report MSR-TR-2013-6. Microsoft Research.

Frank McSherry and Kunal Talwar. 2007. Mechanism Design via Differential Privacy. In *IEEE Symposium on Foundations of Computer Science (FOCS), Providence, Rhode Island*. 94–103. http://doi.ieeecomputersociety.org/10.1109/FOCS.2007.41

George C. Necula. 2000. Translation validation for an optimizing compiler. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Vancouver, British Columbia*, Vol. 35. 83–94.

Federico Olmedo. 2014. *Approximate relational reasoning for probabilistic programs*. Ph.D. Dissertation. Universidad Politécnica de Madrid. http://oa.upm.es/23088/1/FEDERICO_OLMEDO.pdf

Amir Pnueli, Michael Siegel, and Eli Singerman. 1998. Translation validation. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lisbon, Portugal* (1998), 151–166.

Jason Reed and Benjamin C. Pierce. 2010. Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy. In *ACM SIGPLAN International Conference on Functional Programming (ICFP), Baltimore, Maryland*. http://dl.acm.org/citation.cfm?id=1863568

Tetsuya Sato. 2016. Approximate Relational Hoare Logic for Continuous Random Samplings. In *Conference on the Mathematical Foundations of Programming Semantics (MFPS), Pittsburgh, Pennsylvania (Electronic Notes in Theoretical Computer Science)*, Vol. 325. Elsevier, 277–298. https://arxiv.org/abs/1603.01445

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. 2006. Combinatorial sketching for finite programs. In *Asian Symposium on Programming Languages and Systems (APLAS), San Jose, California*. 404–415. https://doi.org/10.1145/1168857.1168907

Marcelo Sousa and Isil Dillig. 2016. Cartesian Hoare Logic for Verifying k-safety Properties. *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Santa Barbara, California*, 57–69.

Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. 2010. From Program Verification to Program Synthesis. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Madrid, Spain*. New York, NY, USA, 313–326. https://doi.org/10.1145/1706299.1706337

Tachio Terauchi and Alexander Aiken. 2005. Secure information flow as a safety problem. In *International Symposium on Static Analysis (SAS), London, England (Lecture Notes in Computer Science)*, Vol. 3672. Springer-Verlag, 352–367.

Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. 2017. A framework for adaptive differential privacy. In *ACM SIGPLAN International Conference on Functional Programming (ICFP), Oxford, England*.

Danfeng Zhang and Daniel Kifer. 2017. LightDP: Towards Automating Differential Privacy Proofs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*. 888–901. https://arxiv.org/abs/1607.08228