View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Differentially Private Bayesian Programming

Citation for published version:
Barthe, G, Farina, GP, Gaboardi, M, Arias, EJG, Gordon, A, Hsu, J & Strub, P-Y 2016, Differentially Private Bayesian Programming. in 23rd ACM Conference on Computer and Communications Security CCS 2016. ACM, pp. 68-79 , 23rd ACM Conference on Computer and Communications Security, Vienna, Austria, 24/10/16. DOI: 10.1145/2976749.2978371

Digital Object Identifier (DOI):
10.1145/2976749.2978371

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
23rd ACM Conference on Computer and Communications Security CCS 2016

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Apr. 2019

Differentially Private Bayesian Programming

Gilles Barthe
IMDEA Software

Gian Pietro Farina
University at Buffalo, SUNY

Marco Gaboardi
University at Buffalo, SUNY

Emilio Jesús Gallego Arias
CRI Mines-ParisTec

Andy Gordon
Microsoft Research, Cambridge UK

Justin Hsu
University of Pennsylvania

Pierre-Yves Strub
IMDEA Software

ABSTRACT

We present PrivInfer, an expressive framework for writing and verifying differentially private Bayesian machine learning algorithms. Programs in PrivInfer are written in a rich functional probabilistic programming language with constructs for performing Bayesian inference. Then, differential privacy of programs is established using a relational refinement type system, in which refinements on probability types are indexed by a metric on distributions. Our framework leverages recent developments in Bayesian inference, probabilistic programming languages, and in relational refinement types. We demonstrate the expressiveness of PrivInfer by verifying privacy for several examples of private Bayesian inference.

1. INTRODUCTION

Differential privacy [17] is emerging as a gold standard in data privacy. Its statistical guarantee ensures that the probability distribution on outputs of a data analysis is almost the same as the distribution on outputs from a hypothetical dataset that differs in one individual. A standard way to ensure differential privacy is by perturbing the data analysis adding some statistical noise. The magnitude and the shape of noise must provide a protection to the influence of an individual on the result of the analysis, while ensuring that the algorithm provides useful results. Two properties of differential privacy are especially relevant for this work: (1) *composability*, (2) the fact that differential privacy works well on large datasets, where the presence or absence of an individual has limited impact. These two properties have led to the design of tools for differentially private data analysis. Many of these tools use programming language techniques to ensure that the resulting programs are indeed differentially private [2–4, 6, 19–21, 29, 33]. Moreover, property (2) has encouraged the interaction of the differential privacy community with the machine learning community to design privacy-preserving machine learning techniques, e.g. [11, 18, 25, 38]. At the same time, researchers in *probabilistic programming* are exploring programming languages as tools for machine learning. For example, concerning Bayesian inference: probabilistic programming allows data analysts to specify the probabilistic model, its parameters, and the data observations in the same format—as a specially crafted program. Given this program, Bayesian inference produces a distribution over the parameters of the model representing our updated beliefs on them. Several works have explored the design of programming languages to compute efficiently the updated beliefs in order to produce efficient and usable tools for machine learning, e.g. [22, 24, 28, 31, 32, 35].

Recently, research in Bayesian inference and machine learning has turned to privacy-preserving Bayesian inference (Dimitrakakis et al. [14], Williams and McSherry [37], Zhang et al. [39], Zheng [40]) where the observed data is private. Bayesian inference is a deterministic process, so releasing the posterior distribution would violate differential privacy. Nerverthless, techniques have been developed to make Bayesian inference differentially private. Basic techniques add noise on the input data, or add noise on the result of the data analysis, while more advanced techniques can ensure differential privacy by taking samples from the posterior. The diversity of such approaches makes Bayesian inference an attractive target for verification tools for differential privacy.

In this work we present PrivInfer, a programming framework combining verification techniques for differential privacy with learning techniques for Bayesian inference in a functional setting. PrivInfer consists of two main components: a probabilistic functional language for Bayesian inference, and a relational higher-order type system that can verify differential privacy for programs written in this language.

The core idea of Bayesian learning is to use conditional distributions to represent the beliefs updated after some observations. PrivInfer, similarly to other programming languages for inference models conditioning on data explicitly. In particular, we extend the functional language **PCF** with an **observe** statement.

Even though, as we said, Bayesian inference's output is a probability distribution, it is still a deterministic process, while ensuring differential privacy involves sampling from some distribution. To handle these two views on distributions PrivInfer distinguishes between *symbolic* distributions and *actual* distributions. The former represent the result of an inference while the latter are used to represent actual random computations, e.g. differentially private computations (*mechanisms*). We parametrize our language with an algorithm to perform Bayesian inference returning symbolic

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CCS '16, October 24–28, 2016, Vienna, Austria.

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4139-4/16/10..

DOI: http://dx.doi.org/XXXX.XXXX

distributions, and mechanisms to ensure differential privacy returning actual distributions.

Differential privacy is a probabilistic 2-property, i.e. a property expressed over pairs of execution traces of the program. Because of this it presents several challenges for formal reasoning. To address these challenges we use an approach based on approximate relational higher-order refinement type system called HOARe^2 [6]. We show how to extend this approach to deal with the constructions that are needed for Bayesian inference like the observe construct and the distinction between symbolic and actual distribution.

Another important aspect of the verification of differential privacy is reasoning about the *sensitivity* of a data analysis. This measures the influence that two databases differing in one individual can have on the output. Calibrating noise to sensitivity ensures that the data analysis provides sufficient privacy. In Bayesian inference, the output of the computation is a distribution (often defined by a few numeric parameters) for which one can consider different measures. A simple approach is to considered standard metrics (Euclidean, Hamming, etc) to measure the distance between the parameters. Another more principled approach is instead to consider distances between distributions—rather than the parameters.

The type system of PrivInfer allows one to reason about the parameters of a distribution, using standard metrics, but also about the distribution itself using f*-divergences*, a class of probability metrics including some well known examples like total variation distance, Hellinger distance, KL divergence, etc. So, overall we extend the approach of PrivInfer in three directions:

- we provide a relational typing rule for observe and for infer,
- we provide a generalization of the relational type system of HOARe^2 to reason about symbolic and actual distributions,
- we generalize the probability polymonad of HOARe^2 to reason about general f-divergence.

The combination of these three contributions allows us to address Bayesian inference, which is not supported by HOARe^2.

To illustrate the different features of our approach we show how different basic Bayesian data analysis can be guaranteed differentially private in three different ways: by adding noise on the input, by adding noise on the parameters with sensitivity measured using the ℓ_1-distance between the parameters, and finally by adding noise on the distributions with sensitivity measured using f-divergences. This shows that PrivInfer can be used for a diverse set of Bayesian data analyses.

Summing up, the contributions of our work are:

- A probabilistic extension $\mathbf{PCF}_p$ of **PCF** for Bayesian inference that serves as the language underlying our framework PrivInfer (§ 4). This includes an observe statement as well as primitives for handling symbolic and actual distributions.
- A higher-order approximate relational type system for reasoning about properties of two runs of programs from PrivInfer (§ 5). In particular, the type system permits to reason about f-divergences. The f-divergences can be used to reason about differential privacy as well as about program sensitivity for Bayesian inference. The relational type system can also reason about symbolic distributions as well.
- We show on several examples how PrivInfer can be used to reason about differential privacy (§ 6). We will explore three ways to guarantee differential privacy: by adding noise on the input, by adding noise on the output parameters based on ℓ_p-norms, and by adding noise on the output parameters based on f-divergences.

2. BAYESIAN INFERENCE AND MOTIVATIONS

Our work is motivated by Bayesian inference, a statistical method which takes a *prior* distribution $\Pr(\xi)$ over a parameter ξ and some observed data x, and produces the *posterior* distribution $\Pr(\xi \mid x)$, an updated version of the prior distribution. Bayesian inference is based on *Bayes' theorem*, which gives a formula for the posterior distribution:

$$\Pr(\xi \mid x) = \frac{\Pr(x \mid \xi) \cdot \Pr(\xi)}{\Pr(x)}$$

The expression $\Pr(x \mid \xi)$ is the *likelihood* of ξ when x is observed. This is a function $\mathcal{L}_x(\xi)$ of the parameter ξ for fixed data x, describing the probability of observing the data x given a specific value of the parameter ξ. Since the data x is considered fixed, the expression $\Pr(x)$ denotes a normalization constant ensuring that $\Pr(\xi \mid x)$ is a probability distribution. The choice of the prior reflects the prior knowledge or belief on the parameter ξ before any observation has been performed. For convenience, the prior and the likelihood are typically chosen in practice so that the posterior belongs to the same family of distributions as the prior. In this case the prior is said to be *conjugate prior* for the likelihood. Using conjugate priors, besides being mathematically convenient in the derivations, ensures that Bayesian inference can be performed by a recursive process over the data.

Our goal is to perform Bayesian inference under differential privacy. We provide the formal definition of differential privacy in Definition 3.1. For the purpose of this section it is enough to keep in mind what we said in the previous section: differential privacy is a statistical guarantee that requires the answer to a data analysis to be statistically close when run on two *adjacent* databases, i.e. databases that differ in one individual. In the *vanilla* version of differential privacy, the notion of "statistically close" is measured by a parameter ϵ. A way to achieve differential privacy is by adding some *statistical noise* and we present several primitives for doing this in § 3. We will use here one of them, the *exponential mechanism*, denoted $\mathsf{ExpMech}_\epsilon$, which returns a possible output with probability proportional to a quality score function Q. Q takes in input a database and a potential output for the statistic computed on the database and gives in output a score representing how *good* that output is for that database. The privacy and the utility of the mechanism depend on ϵ and on the *sensitivity* of the quality score function, i.e. a measure of how much the quality score can differ for two adjacent databases.

As a motivating example we will consider a simple Bayesian task: learning the bias of a coin from some observations. To give it more context, we can think of the observations as medical records asserting whether patients from a sample population have a disease or not. We can perform Bayesian

learning to establish how likely it is to have the disease in the population. We will show how to program this simple example and make it differentially private in PrivInfer.

First, the input of this example is a set of binary observations describing whether any given patient has the disease. We consider this the *private* information that we want to protect. We also assume that the number of patient n is *public* and that the *adjacency* condition for differential privacy states that two databases differ in the data of one patient. In our concrete case this means that two databases d, d' are adjacent if all of their records are the same except for one record that is 0 in one database and 1 in the other.

While in abstract our problem can be described as estimating the bias of a coin, we need to be more formal and provide the precise model and the parameters that we want to estimate. We can incorporate our initial belief on the fairness of the coin using a prior distribution on the bias ξ given by a *beta distribution*. This is a distribution over $[0, 1]$ with probability density:

$$\mathsf{beta}(\xi \mid a, b) = \frac{\xi^{a-1}(1-\xi^{b-1})}{B(a,b)}$$

where $a, b \in \mathbb{R}^+$ are parameters and B denotes the beta function. The likelihood is the probability that a series of i.i.d samples from a Bernoulli distributed random variable with bias ξ matches the observations. Using an informal notation[1] we can write the following program in PrivInfer:

$$\mathsf{infer}\Big(\mathsf{observe}\ \big(\lambda r.\mathsf{bernoulli}(r) = \mathsf{obs}\big)\ \mathsf{beta}(a,b)\Big) \qquad (1)$$

The term infer represents an inference algorithm and the observe statement is used to describe the model. Expression (1) denotes the posterior distribution that is computed using Bayes' theorem with prior $\mathsf{beta}(a, b)$ and with likelihood $(\lambda r.\mathsf{bernoulli}(r) = \mathsf{obs})$.

Now, we want to ensure differential privacy. We have several ways to proceed. A first natural idea is to perturbate the input data using the exponential mechanism. The fact that differential privacy is closed under post-processing ensures that this is enough to obtain differential privacy for the whole program. This gives a program as follows:

$$\mathsf{infer}\Big(\mathsf{observe}(\lambda r.\mathsf{bernoulli}(r) = \mathsf{ExpMech}_\epsilon\ \mathsf{Q}\ \mathsf{obs})\ \mathsf{beta}(a,b)\Big)$$

In the notation above we denoted by Q the scoring function. Since obs is a boolean, we can use a quality score function that gives score 1 to b if $b = \mathsf{obs}$ and 0 otherwise. This function has sensitivity 1 and so one achieves $(\epsilon, 0)$-differential privacy. This is a very simple approach, but in some situations it can already be very useful [37].

Another intuitive approach is the one of adding noise on the output. In this case the output is the posterior which is a $\mathsf{beta}(a', b')$ for some values a', b'. Using again the exponential mechanism we can describe it as

$$\mathsf{ExpMech}_\epsilon\ \mathsf{Q}\Big(\mathsf{infer}\big(\mathsf{observe}(\lambda r.\mathsf{bernoulli}(r) = \mathsf{obs})\mathsf{beta}(a,b)\big)\Big)$$

In this case, however the exponential mechanism is not applied to booleans but instead to distributions. In particular, to distributions of the shape $\mathsf{beta}(a', b')$. So, a natural question is which Q we can use as quality score function and what is its sensitivity in this case.

There are two natural choices. The first one is to consider (a', b') as a vector and measuring the possible distance in term of some metric on vectors, e.g. the one given by ℓ_1 norm $d((a, b), (a', b')) = |a - a'| + |b - b'|$. The second is to consider $\mathsf{beta}(a', b')$ as an actual distribution and then use some statistical distance, e.g. use Hellinger distance $\Delta_{\mathcal{H}}(\mathsf{beta}(a, b), \mathsf{beta}(a', b'))$. The two approaches give different sensitivity and have different utility. The utility also depend on the metrics that we use to estimate it [39]. So, it is important that our system PrivInfer can prove privacy for both the approaches.

3. BACKGROUND

3.1 Probability and Distributions

In our work we will consider discrete distributions. Following Dwork and Roth [16] we will use standard names for several continuous distributions but we will consider them to be the approximate discrete versions of these distributions up to arbitrary precision.

We define the set $\mathcal{D}(A)$ of *distributions* over a set A as the set of functions $\mu : A \to [0, 1]$ with discrete $\mathsf{support}(\mu) = \{x \mid \mu\, x \neq 0\}$, such that $\sum_{x \in A} \mu\, x = 1$. In our language we will consider only distribution over basic types, this guarantees that all our distributions are discrete (see § 4).

We will use several basic distributions like uniform, bernoulli, normal, beta, etc. These are all standard distributions and we omit their definition here. We will also use some notation to describe distributions. For instance, given an element $a \in A$, we will denote by $\mathbb{1}_a$ the probability distribution that assigns all mass to the value a. We will also denote by $\mathsf{bind}\ \mu\ M$ the composition of a distribution μ over the set A with a function M that takes a value in A and returns a distribution over the set B.

3.2 Differential Privacy

Differential privacy is a strong, quantitative notion of statistical privacy proposed by Dwork et al. [17]. In the standard setting, we consider a program (sometimes called a *mechanism*) that takes a *private database* d as input, and produces a distribution over outputs. Intuitively, d represents a collection of data from different individuals. When two databases d, d' are identical except for a single individual's record, we say that d and d' are *adjacent*[2], and we write $d\ \Phi\ d'$. Then, differential privacy states that the output distributions from running the program on two adjacent databases should be statistically similar. More formally:

Definition 3.1 (Dwork et al. [17])**.** Let $\epsilon, \delta > 0$ be two numeric parameters, let D be the set of databases, and let R be the set of possible outputs. A program $M : D \to \mathcal{D}(R)$ satisfies (ϵ, δ)-*differential privacy* if

$$\Pr(M(d) \in S) \leqslant e^\epsilon \Pr(M(d') \in S) + \delta$$

for all pairs of adjacent databases $d, d' \in D$ such that $d\ \Phi\ d'$, and for every subset of outputs $S \subseteq R$.

As shown by Barthe et al. [3], we can reformulate differential privacy using a specific statistical ϵ-distance ϵ-D:

[1] We omit in particular the monadic probabilistic constructions. A formal description of this example can be found in § 6.

[2] In our concrete examples we will consider sometime as *adjacent* also two databases that differ by *at most* one individual.

Lemma 3.1. *Let $\epsilon, \delta \in \mathbb{R}^+$. Let D be the set of databases, and let R be the set of possible outputs. A program $M : D \to \mathcal{D}(R)$ satisfies (ϵ, δ)-*differential privacy *iff $\epsilon\text{-}\mathsf{D}(M(d), M(d')) \leqslant \delta$, where d, d' are adjacent databases and*

$$\epsilon\text{-}\mathsf{D}(\mu_1, \mu_2) \equiv \max_{E \subseteq R} \big(\Pr_{x \leftarrow \mu_1}[x \in E] - e^{\epsilon} \cdot \Pr_{x \leftarrow \mu_2}[x \in E] \big)$$

for $\mu_1, \mu_2 \in \mathcal{D}(R)$.

Differential privacy is an unusually robust notion of privacy. It degrades smoothly when private mechanisms are composed in sequence or in parallel, and it is preserved under any post-processing that does not depend on the private database. The following lemmas capture these properties:

Lemma 3.2 (Post-processing)**.** *Let $M : D \to \mathcal{D}(R)$ be an (ϵ, δ)-differentially private program. Let $N : R \to \mathcal{D}(R')$ be an arbitrary randomized program. Then $\lambda d.\mathsf{bind}\,(M\,d)\,N : D \to \mathcal{D}(R')$ is (ϵ, δ)-differentially private.*

Differential privacy enjoys different composition schemes, we report here one of the simpler and most used.

Lemma 3.3 (Composition)**.** *Let $M_1 : D \to \mathcal{D}(R_1)$, and $M_2 : D \to \mathcal{D}(R_2)$ respectively (ϵ_1, δ_1) and (ϵ_2, δ_2) differentially private programs. Let $M : D \to \mathcal{D}(R_1 \times R_2)$ the program defined as $M(x) \equiv (M_1(x), M_2(x))$. Then, M is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$ differentially private.*

Accordingly, complex differentially private programs can be easily assembled from simpler private components, and researchers have proposed a staggering variety of private algorithms which we cannot hope to summarize here. (Interested readers can consult Dwork and Roth [16] for a textbook treatment.)

While these algorithms serve many different purposes, the vast majority are constructed from just three private operations, which we call *primitives*. These primitives offer different ways to create private mechanisms from non-private functions. Crucially, the function must satisfy the following *sensitivity* property:

Definition 3.2. Let $k \in \mathbb{R}^+$. Suppose $f : A \to B$ is a function, where A and B are equipped with distances d_A and d_B. Then f is *k-sensitive* if

$$d_B(f(a), f(a')) \leqslant k \cdot d_A(a, a')$$

for every $a, a' \in A$.

Intuitively, k-sensitivity bounds the effect of a small change in the input, a property that is similar in spirit to the differential privacy guarantee. With this property in hand, we can describe the three basic primitive operations in differential privacy, named after their noise distributions.

The Laplace mechanism. The first primitive is the standard way to construct a private version of a function that maps databases to numbers. Such functions are also called *numeric queries*, and are fundamental tools for statistical analysis. For instance, the function that computes the average age of all the individuals in a database is a numeric query. When the numeric query has bounded sensitivity, we can use the *Laplace mechanism* to guarantee differential privacy.

Definition 3.3. Let $\epsilon \in \mathbb{R}^+$ and let $f : D \to \mathbb{R}$ be a numeric query. Then, the *Laplace mechanism* maps a database $d \in D$ to $f(d) + \nu$, where ν is a drawn form the Laplace distribution with scale $1/\epsilon$. This distribution has the following probability density function:

$$\mathrm{Lap}_{1/\epsilon}(x) = \frac{\epsilon}{2} \exp(-|x|\epsilon).$$

If f is a k-sensitive function, then the Laplace mechanism is $(k\epsilon, 0)$-differentially private.

The Gaussian mechanism. The Gaussian mechanism is an alternative to the Laplace mechanism, adding Gaussian noise with an appropriate standard deviation to release a numeric query. Unlike the Laplace mechanism, the Gaussian mechanism does not satisfy $(\epsilon, 0)$-privacy for any ϵ. However, it satisfies (ϵ, δ)-differential privacy for $\delta \in \mathbb{R}^+$.

Definition 3.4. Let $\epsilon, \delta \in \mathbb{R}$ and let $f : D \to \mathbb{R}$ be a numeric query. Then, the *Gaussian mechanism* maps a database $d \in D$ to $f(d) + \nu$, where ν is a drawn from the Gaussian distribution with standard deviation

$$\sigma(\epsilon, \delta) = \sqrt{2 \ln(1.25/\delta)}/\epsilon.$$

If f is a k-sensitive function for $k < 1/\epsilon$, then the Gaussian mechanism is $(k\epsilon, \delta)$-differentially private.

The exponential mechanism. The first two primitives can make numeric queries private, but in many situations we may want to privately release a non-numeric value. To accomplish this goal, the typical tool is the *exponential mechanism* [30], our final primitive. This mechanism is parameterized by a set R, representing the range of possible outputs, and a *quality score* function $q : D \times R \to \mathbb{R}$, assigning a real-valued score to each possible output given a database.

The exponential mechanism releases an output $r \in R$ with approximately the largest quality score on the private database. The level of privacy depends on the sensitivity of q in the database. Formally:

Definition 3.5 (McSherry and Talwar [30])**.** Let $\epsilon \in \mathbb{R}^+$. Let R be the set of outputs, and $q : D \times R \to \mathbb{R}$ be the quality score. Then, the *exponential mechanism* on database $d \in D$ releases $r \in R$ with probability proportional to

$$\Pr(r) \sim \exp\left(\frac{q(d, r)\epsilon}{2}\right)$$

If f is a k-sensitive function in d for any fixed $r \in R$, then the exponential mechanism is $(k\epsilon, 0)$-differentially private.

3.3 f-divergences

As we have seen, differential privacy is closely related to function sensitivity. To verify differential privacy for the result of probabilistic inferences, we will need to work with several notions of distance between distributions. These distances can be neatly described as *f-divergences* [12], a rich class of metrics on probability distributions. Inspired by the definition of relative entropy, f-divergences are defined by a convex function f. Formally:

Definition 3.6 (Csiszár and Shields [13])**.** Let $f(x)$ be a convex function defined for $x > 0$, with $f(1) = 0$. Let μ_1, μ_2 two distributions over A. Then, the *f-divergence* of μ_1 from μ_2, denoted $\Delta_f(\mu_1 \mid \mu_2)$ is defined as:

$$\Delta_f(\mu_1 \mid \mu_2) = \sum_{a \in A} \mu_2(a) f\Big(\frac{\mu_1(a)}{\mu_2(a)}\Big)$$

f-diverg.	$f(x)$	Simplified form
$\mathsf{SD}(x)$	$\frac{1}{2}\ \lvert x-1\rvert$	$\sum_{a\in A}\frac{1}{2}\lvert\mu_1(a)-\mu_2(a)\rvert$
$\mathsf{HD}(x)$	$\frac{1}{2}\ (\sqrt{x}-1)^2$	$\sum_{a\in A}\frac{1}{2}\left(\sqrt{\mu_1(a)}-\sqrt{\mu_2(a)}\right)^2$
$\mathsf{KL}(x)$	$x\ln(x)-x+1$	$\sum_{a\in A}\mu_1(a)\ln\left(\frac{\mu_1(a)}{\mu_2(a)}\right)$
ϵ-$\mathsf{D}(x)$	$\max(x-e^\epsilon,0)$	$\sum_{a\in A}\max\left(\mu_1(a)-e^\epsilon\mu_2(a),0\right)$

Table 1: f-divergences for statistical distance (SD), Hellinger distance (HD), KL divergence (KL), and ϵ-distance (ϵ-D)

where we assume $0\cdot f(\frac{0}{0})=0$ and

$$0\cdot f\left(\frac{a}{0}\right)=\lim_{t\to 0}t\cdot f\left(\frac{a}{t}\right)=a\lim_{u\to\infty}\left(\frac{f(u)}{u}\right).$$

If $\Delta_f(\mu_1\ \ |\ \mu_2)\leqslant\delta$ we say that μ_1 and μ_2 are (f,δ)-close.

Examples of f-divergences include *KL-divergence*, *Hellinger* distance, and *total variation* distance. Moreover, Barthe and Olmedo [2] showed how the *ϵ-distance* of Lemma 3.1 can be seen as an f-divergence for differential privacy. These f-divergences are summarized in Table 1. Notice that some of the f-divergences in the table above are not symmetric. In particular, this is the case for KL-divergence and ϵ-distance, which we use to describe (ϵ,δ)-differential privacy. We will denote by $\mathcal{F}$ the class of functions meeting the requirements of Definition 3.6.

Not only do f-divergences measure useful statistical quantities, they also enjoy several properties that are useful for formal verification. (e.g. see [13]). A property that is worth mentioning and that will be used implicitly in our example is the following.

Theorem 3.1 (Data processing inequality)**.** *Let $f\in\mathcal{F}$, μ_1,μ_2 be two distributions over A, and M be a function (potentially randomized) mapping values in A to distributions over B. Then, we have:*

$$\Delta_f(\mathsf{bind}\,\mu_1\,M,\mathsf{bind}\,\mu_2\,M)\leqslant\Delta_f(\mu_1,\mu_2)$$

Another important property for our framework is composition. As shown by Barthe and Olmedo [2] we can compose f-divergences in an additive way. More specifically, they give the following definition.

Definition 3.7 (Barthe and Olmedo [2])**.** Let $f_1,f_2,f_3\in\mathcal{F}$. We say that (f_1,f_2) are f_3 composable if and only if for every A,B, two distributions μ_1,μ_2 over A, and two functions M_1,M_2 mapping values in A to distributions over B we have

$$\Delta_{f_3}(\mathsf{bind}\,\mu_1\,M_1,\mathsf{bind}\,\mu_2\,M_2)\leqslant\Delta_{f_1}(\mu_1,\mu_2)+\sup_v\Delta_{f_2}(M_1\,v,M_2\,v)$$

In particular, we have the following.

Lemma 3.4 (Barthe and Olmedo [2])**.**

- $(\epsilon_1$-D,ϵ_2-$\mathsf{D})$ *are* $(\epsilon_1+\epsilon_2)$-DP *composable.*
- $(\mathsf{SD},\mathsf{SD})$ *are* SD *composable.*
- $(\mathsf{HD},\mathsf{HD})$ *are* HD *composable.*
- $(\mathsf{KL},\mathsf{KL})$ *are* KL *composable.*

This form of composition will be internalized by the relational refinement type system that we will present in § 5.

4. PrivInfer

The main components of PrivInfer are a language that permits to express Bayesian inference models and a type system for reasoning in a relational way about programs from the language.

4.1 The language

The language underlying PrivInfer is a probabilistic programming extension of **PCF** that we will call $\mathbf{PCF}_p$. Expressions of $\mathbf{PCF}_p$ are defined by the following grammar

$$\begin{array}{rcl} e & ::= & x\mid c\mid e\ e\mid\lambda x.\,e\\ & \mid & \texttt{letrec}\ f\ x=e\mid\texttt{case}\ e\ \texttt{with}\ [d_i\ \overline{x_i}\Rightarrow e_i]_i\\ & \mid & \texttt{return}\ e\mid\texttt{mlet}\ x=e\ \texttt{in}\ e\\ & \mid & \mathsf{observe}\,x\Rightarrow e\,\mathsf{in}\,e\mid\mathsf{infer}(e)\mid\mathsf{ran}(e)\\ & \mid & \mathsf{bernoulli}(e)\mid\mathsf{normal}(e,e)\mid\mathsf{beta}(e,e)\mid\mathsf{uniform}()\\ & \mid & \mathsf{lapMech}(e,e)\mid\mathsf{gaussMech}(e,e)\mid\mathsf{expMech}(e,e,e)\end{array}$$

where c represent a constant from a set $\mathcal{C}$ and x a variable. We will denote by $\mathbf{PCF}_p\ (\mathcal{X})$ the set of expression of PrivInfer where the variables are taken from the set $\mathcal{X}$.

We will consider only expressions that are well typed using simple types of the form

$$\begin{array}{rcl}\tau,\sigma & ::= & \widetilde{\tau}\mid\mathfrak{M}[\widetilde{\tau}]\mid\mathfrak{M}[\mathfrak{D}[\widetilde{\tau}]]\mid\mathfrak{D}[\widetilde{\tau}]\mid\tau\to\sigma\\ \widetilde{\tau} & ::= & \bullet\mid\mathbb{B}\mid\mathbb{N}\mid\mathbb{R}\mid\mathbb{R}^+\mid\overline{\mathbb{R}}^+\mid[0,1]\mid\widetilde{\tau}\ \texttt{list}.\end{array}$$

where $\widetilde{\tau}$ are basic types. As usual a typing judgment is a judgment of the shape $\Gamma\vdash e:\tau$ where an environment Γ is an assignment of types to variables. The simply typed system of PrivInfer is an extension of the one in Barthe et al. [6]; in Figure 1 we only present the rules specific to PrivInfer.

The syntax and types of $\mathbf{PCF}_p$ extends the one of **PCF** by means of several constructors. Basic types include the unit type $\bullet$ and types for booleans $\mathbb{B}$ and natural numbers $\mathbb{N}$. We also have types for real numbers $\mathbb{R}$, positive real numbers $\mathbb{R}^+$, positive real number plus infinity $\overline{\mathbb{R}}^+$ and for real numbers in the unit interval $[0,1]$. Finally we have lists over basic types. Simple types combines basic types using arrow types, a probability monad $\mathfrak{M}[\widetilde{\tau}]$ over the basic type $\widetilde{\tau}$, and a type $\mathfrak{D}[\widetilde{\tau}]$ representing *symbolic distributions* over the basic type $\widetilde{\tau}$. The probability monad can also be over symbolic distributions. Probabilities (*actual distributions*) are encapsulated in the probabilistic monad $\mathfrak{M}[\widetilde{\tau}]$ that can be manipulated by the let-binder $\texttt{mlet}\ x=e_1\ \texttt{in}\ e_2$ and by the unit $\texttt{return}\ e$. Symbolic distributions are built using basic probabilistic primitives like $\mathsf{bernoulli}(e)$ for Bernoulli distributions, $\mathsf{normal}(e_1,e_2)$ for normal distribution, etc. These primitives are assigned types as described in Figure 2. For symbolic distributions we also assume that we have an operation $\mathsf{getParams}$ to extract the parameters. We also have primitives $\mathsf{lapMech}(e_1,e_2)$, $\mathsf{gaussMech}(e_1,e_2)$ and $\mathsf{expMech}(e_1,e_2,e_3)$ that provide implementations for the mechanism ensuring differential privacy as described in § 3.2.

Finally, we have three special constructs for representing learning. The primitive $\mathsf{observe}\,x\Rightarrow e_1\,\mathsf{in}\,e_2$ can be used to describe conditional distributions. This is a functional version of a similar primitive used in languages like Fun [23]. This primitive takes two arguments, a prior e_2 and a predicate e_1 over x. The intended semantics is the one provided by Bayes' theorem: it filters the prior by means of the observation provided by e_1 and renormalize the obtained distribution (see Section § 4.2 for more details). The primitives $\mathsf{infer}(e)$ and $\mathsf{ran}(e)$ are used to transform symbolic distributions in

$$\textsc{BindM}\ \frac{\Gamma \vdash e_1 : \mathfrak{M}[T_1] \qquad \Gamma, x : T_1 \vdash e_2 : \mathfrak{M}[T_2]}{\Gamma \vdash \texttt{mlet}\ x = e_1\ \texttt{in}\ e_2 : \mathfrak{M}[T_2]} \qquad \textsc{UnitM}\ \frac{\vdash \mathfrak{M}[T] \qquad \Gamma \vdash e : T}{\Gamma \vdash \texttt{return}\ e : \mathfrak{M}[T]}$$

$$\textsc{Observe}\ \frac{\Gamma \vdash e_1 : \mathfrak{M}[\widetilde{\tau}]\Gamma, x : \widetilde{\tau} \vdash e_2 : \mathfrak{M}[\mathbb{B}]}{\Gamma \vdash \mathsf{observe}\, x \Rightarrow e_2\, \mathsf{in}\, e_1 : \mathfrak{M}[\widetilde{\tau}]} \qquad \textsc{Ran}\ \frac{\Gamma \vdash e : \mathfrak{D}[\widetilde{\tau}]}{\Gamma \vdash \mathsf{ran}(e) : \mathfrak{M}[\widetilde{\tau}]} \qquad \textsc{Infer}\ \frac{\Gamma \vdash e : \mathfrak{M}[\widetilde{\tau}]}{\Gamma \vdash \mathsf{infer}(e) : \mathfrak{D}[\widetilde{\tau}]}$$

Figure 1: $\mathbf{PCF}_p$ type system (selected rules)

$\mathsf{uniform} : \mathfrak{D}[[0,1]]$
$\mathsf{bernoulli} : [0,1] \to \mathfrak{D}[\mathbb{B}]$
$\mathsf{beta} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathfrak{D}[[0,1]]$
$\mathsf{normal} : \mathbb{R} \times \mathbb{R}^+ \to \mathfrak{D}[\mathbb{R}]$
$\mathsf{lapMech} : \mathbb{R}^+ \times \mathbb{R} \to \mathfrak{M}[\mathbb{R}]$
$\mathsf{gaussMech} : \mathbb{R}^+ \times \mathbb{R} \to \mathfrak{M}[\mathbb{R}]$
$\mathsf{expMech} : \mathbb{R} \times ((D,R) \to \mathbb{R}) \times D \to \mathfrak{M}[R]$

Figure 2: Primitive distributions types.

actual distributions and viceversa. In particular, $\mathsf{infer}(e)$ is the main component performing probabilistic inference.

4.2 Denotational Semantics

The semantics of $\mathbf{PCF}_p$ is largely standard. Basic types are interpreted in the corresponding sets, e.g. $[\![\bullet]\!] = \{\bullet\}$, $[\![\mathbb{B}]\!] = \{\mathsf{true}, \mathsf{false}\}$, $[\![\mathbb{N}]\!] = \{0, 1, 2, \ldots\}$, etc. As usual, arrow types $[\![\tau \to \sigma]\!]$ are interpreted as set of functions $[\![\tau]\!] \to [\![\sigma]\!]$. A monadic type $\mathfrak{M}[\tau]$ for $\tau \in \{\widetilde{\tau}, \mathfrak{D}[\widetilde{\tau}]\}$ is interpreted as the set of discrete probabilities over τ, i.e.:

$$[\![\mathfrak{M}[\tau]]\!] = \left\{\mu : [\![\tau]\!] \to \mathbb{R}^+ \mid \mathsf{supp}(\mu)\, \texttt{discrete} \wedge \sum_{x \in [\![\tau]\!]} \mu\, x = 1\right\}$$

Types of the shape $\mathfrak{D}[\widetilde{\tau}]$ are interpreted in set of symbolic representations for distributions parametrized by values. As an example, $\mathfrak{D}[\mathbb{B}]$ is interpreted as:

$$[\![\mathfrak{D}[\mathbb{B}]]\!] = \left\{\mathsf{bernoulli}(v) \mid v \in [0,1]\right\}$$

The interpretation of expressions is given as usual under a validation θ which is a finite map from variables to values in the interpretation of types. We will say that θ validates an environment Γ if $\forall x : \tau \in \Gamma$ we have $\theta(x) \in [\![\tau]\!]$. For most of the expressions the interpretation is standard, we detail the less standard interpretations in Figure 3. Probabilistic expressions are interpreted into discrete probabilities. In particular, $[\![\texttt{return}\ e]\!]_\theta$ is defined as the Dirac distribution returning $[\![e]\!]_\theta$ with probability one. The mlet-binder $\texttt{mlet}\ x = e_1\ \texttt{in}\ e_2$ composes probabilities. The expression $\mathsf{observe}\, x \Rightarrow t\, \mathsf{in}\, u$ filters the distribution $[\![u]\!]_\theta$ using the predicate $x \Rightarrow t$ and rescales it in order to obtain a distribution. The observe is the key component to have conditional distributions and to update a prior using Bayes' theorem. The semantics of infer relies on a given algorithm[3] AlgInf for inference. We leave the algorithm unspecified because it is not central to our verification task. Symbolic distributions are syntactic constructions, this is reflected in their interpretation. For an example, we give in Figure 3 the interpretation $[\![\mathsf{bernoulli}(e)]\!]_\theta$. The operator ran turns a symbolic distribution in an actual distribution. Its semantics is defined by cases on the given symbolic distribution. In Figure 3 we give its interpretation for the case when the given symbolic distribution is $\mathsf{bernoulli}(e)$. The cases for the other symbolic distributions are similar.

The soundness of the semantics is given by the following:

Lemma 4.1. *If $\Gamma \vdash e : \tau$ and θ validates Γ, then $[\![e]\!]_\theta \in [\![\tau]\!]$.*

[3]In this work we consider only exact inference, and we leave for future works to consider approximate inference. We also consider only programs with a well defined semantics: e.g. we don't consider programs where we make observations of events with 0 probability. We will finally restrict ourselves to inference algorithms that never fail: we could easily simulate this by using the *maybe* monad.

5. RELATIONAL TYPE SYSTEM

5.1 Relational Typing

To reason about differential privacy as well as about f-divergences we will consider a higher-order relational refinement type system. We follow the approach proposed by Barthe et al. [6].

We will distinguish two sets of variables: $\mathcal{X}_\mathcal{R}$ (*relational*) and $\mathcal{X}_\mathcal{P}$ (*plain*). Associated with every relational variable $x \in \mathcal{X}_\mathcal{R}$, we have a left instance $x_\triangleleft$ and a right instance $x_\triangleright$. We write ${\mathcal{X}_\mathcal{R}}^\bowtie$ for $\bigcup_{x \in \mathcal{X}_\mathcal{R}} \{x_\triangleleft, x_\triangleright\}$ and $\mathcal{X}^\bowtie$ for ${\mathcal{X}_\mathcal{R}}^\bowtie \cup \mathcal{X}_\mathcal{P}$.

The set of PrivInfer *expressions* $\mathcal{E}$ is the set of expressions defined over plain variables, i.e. expressions in $\mathbf{PCF}_p(\mathcal{X}_\mathcal{P})$. The set of PrivInfer *relational expressions* $\mathcal{E}^\bowtie$ is the set of expressions defined over plain and relational variables, expressions in $\mathbf{PCF}_p(\mathcal{X}^\bowtie)$, where only non-relational variables can be bound.

The sets of *relational types* $\mathcal{T} = \{T, U, \ldots\}$ and *assertions* $\mathcal{A} = \{\phi, \psi, \ldots\}$ are defined by the following grammars:

$$\begin{array}{rcll}
T, U \in \mathcal{T} & ::= & \widetilde{\tau} \mid \mathfrak{M}_{f,\delta}[\{x :: \widetilde{\tau} \mid \phi\}] \mid \mathfrak{M}_{f,\delta}[\{x :: \mathfrak{D}[\widetilde{\tau}] \mid \phi\}] & \\
 & & \mid \mathfrak{D}[\widetilde{\tau}] \mid \Pi(x :: T).\, T \mid \{x :: T \mid \phi\} & \\
\phi, \psi \in \mathcal{A} & ::= & \mathcal{Q}\ (x : \tau).\, \phi & (x \in \mathcal{X}_\mathcal{P}) \\
 & & \mathcal{Q}\ (x :: T).\, \phi & (x \in \mathcal{X}_\mathcal{R}) \\
 & & \mid \Delta_f^\mathfrak{D}(e^\bowtie, e^\bowtie) \leqslant \delta \mid f \in \mathcal{F} & \\
 & & \mid e^\bowtie = e^\bowtie \mid e^\bowtie \leqslant e^\bowtie \mid \mathcal{C}(\phi_1, \ldots, \phi_n) & \\
\mathcal{C} & = & \{\top/_0, \bot/_0, \neg/_1, \vee/_2, \wedge/_2, \Rightarrow/_2\}, &
\end{array}$$

where $f, \delta, e^\bowtie \in \mathcal{E}^\bowtie$, and $\mathcal{Q} \in \{\forall, \exists\}$.

Relational types extend simple types by means of relational refinements of the shape $\{x :: T \mid \phi\}$. This is a refinement type that uses a relational assertion ϕ stating some relational property that the inhabitants of this type have to satisfy. Relational assertions are first order formulas over some basic predicates: $\Delta_f^\mathfrak{D}(e^\bowtie, e^\bowtie) \leqslant \delta$ asserting a bound on a specific f-divergence, $f \in \mathcal{F}$ asserting that f is a convex function meeting the requirements of Definition 3.6, and $e^\bowtie = e^\bowtie$ and $e^\bowtie \leqslant e^\bowtie$ for the equality and inequality of relational expressions, respectively. Relational types

$$[\![\Gamma \vdash \texttt{return}\ e : \mathfrak{M}[\tau]]\!]_\theta = \mathbb{1}_{[\![e]\!]_\theta} \qquad [\![\Gamma \vdash \texttt{mlet}\ x = e_1\ \texttt{in}\ e_2 : \mathfrak{M}[\sigma]]\!]_\theta = d \mapsto \sum_{g \in [\![\tau]\!]_\theta} ([\![e_1]\!]_\theta(g) \cdot [\![e_2]\!]_{\theta^g_x}(d))$$

$$[\![\Gamma \vdash \mathsf{observe}\, x \Rightarrow t\, \mathsf{in}\, u : \mathfrak{M}[\tau]]\!]_\theta = d \mapsto \frac{[\![u]\!]_\theta(d) \cdot ([\![t]\!]_{\theta^d_x}(\mathsf{true}))}{\sum_{g \in [\![\tau]\!]_\theta} ([\![u]\!]_\theta(g) \cdot ([\![t]\!]_{\theta^g_x}(\mathsf{true})))} \qquad [\![\Gamma \vdash \mathsf{infer}\, e : \mathfrak{D}[\tau]]\!]_\theta = \mathsf{AlgInf}\, e$$

$$[\![\Gamma \vdash \mathsf{bernoulli}(e) : \mathfrak{D}[\mathbb{B}]]\!]_\theta = \mathsf{bernoulli}([\![e]\!]_\theta) \qquad [\![\Gamma \vdash \mathsf{ran}(\mathsf{bernoulli}(e)) : \mathfrak{M}[\mathbb{B}]]\!]_\theta = d \mapsto \begin{cases} [\![e]\!]_\theta & \text{if } d = \mathsf{true} \\ 1 - [\![e]\!]_\theta & \text{otherwise} \end{cases}$$

Figure 3: Interpretation for some of $\mathbf{PCF}_p$ expressions (selected rules).

also refines the probability monad. This has now the shape $\mathfrak{M}_{f,\delta}[\{x :: T \mid \phi\}]$, for $T \in \{\widetilde{\tau}, \mathfrak{D}[\widetilde{\tau}]\}$, and it corresponds to a polymonad [26] or parametric effect monads [27] where f and δ are parameters useful to reason about f-divergences. Relational expressions can appear in relational refinements and in the parameters of the probabilistic monad, so the usual arrow type constructor is replaced by the dependent type constructor $\Pi(x :: T).\, S$.

Before introducing the typing rule of PrivInfer we need to introduce some notation. A relational environment $\mathcal{G}$ is a finite sequence of bindings $(x :: T)$ such that each variable x is never bound twice and only relational variables from $\mathcal{X}_\mathcal{R}$ are bound. We write $x\mathcal{G}$ for the type of x in $\mathcal{G}$. We will denote by $|\cdot|$ the type erasure from relational types to simple types and its extension to environments. We will use instead the notation $\|\cdot\|$, to describe the following map from relational environments to environments $x_\mathfrak{s}\|\mathcal{G}\| = x|\mathcal{G}|$ iff $x \in \mathrm{dom}(\mathcal{G})$, where $\mathfrak{s} \in \{\lhd, \rhd\}$. For example, given a relational binding $(x :: T)$, we have $\|(x :: T)\| = x_\lhd : |T|, x_\rhd : |T|$.

We can now present our relational type system. PrivInfer proves typing judgment of the form $\mathcal{G} \vdash e_1 \sim e_2 :: T$. We will use $\Gamma \vdash e :: T$ as a shorthand for $\Gamma \vdash e \sim e :: T$. Several of the typing rules of PrivInfer come from the system proposed by Barthe et al. [6]. We report some of them in Figure 5. We also extend the subtyping relation of Barthe et al. [6] with the rule for monadic subtyping in Figure 4.

We present the rules that are specific to PrivInfer in Figure 5. The rules UnitM and BindM correspond to the unit and the composition of the probabilistic polymonad. These are similar to the usual rule for the unit and composition of monads but additionally they require the indices to well behave. In particular (f_1, f_2) are required to be f_3 composable to guarantee that the composition is respected. The rules Infer and Ran are similar to their simply typed version but they also transfer the information on the f-divergence from the indices to the refinements and viceversa. The rule Observe requires that the sub-expressions are well typed relationally and it further requires the validity of the assertion:

$$\|\mathcal{G}\| \vdash \Delta_f(\mathsf{observe}\, x_\lhd \Rightarrow e_\lhd\, \mathsf{in}\, e'_\lhd, \mathsf{observe}\, x_\rhd \Rightarrow e_\rhd\, \mathsf{in}\, e'_\rhd) \leqslant \delta''$$

for the δ'' that can then be used as a bound for the f-divergence in the conclusion. This assertion may be surprising at first, since it doesn't consider the bounds δ and δ' for the sub-expressions but instead requires to provide directly a bound for the conclusion—it is not really compositional. The reason for this presentation is that a generic bound in term of δ and δ' would be often too large to say something useful about the conclusion. In fact, estimating bounds on the f-divergence of conditional distributions is a hard task and often it is only expressed in term of prior perturbations [15]. So, instead we prefer to postpone the task to verify a bound to the concrete application where a tighter bound can be found with some calculation. We will see some uses of this rule in the examples in § 6.

$$\text{S-M}\ \frac{\mathcal{G} \vdash T \preceq U \qquad \|\mathcal{G}\| \vdash \delta_i : \overline{\mathbb{R}}^+ \qquad \|\mathcal{G}\| \vdash f_i \in \mathcal{F} \qquad \forall \theta.\, \theta \Vdash \mathcal{G}, x :: T \Rightarrow [\![f_1 \leqslant f_2 \wedge \delta_1 \leqslant \delta_2 < \infty]\!]_\theta}{\mathcal{G} \vdash \mathfrak{M}_{f_1,\delta_1}[T] \preceq \mathfrak{M}_{f_2,\delta_2}[U]}$$

Figure 4: Relational Subtyping (rule for monadic subtyping)

5.2 Relational Interpretation

We want now to give a relational interpretation of relational types so that we can prove the soundness of the relational type system of PrivInfer. Before doing this we need to introduce an important component of our interpretation, the notion of (f, δ)-lifting of a relation, inspired from the relational lifting of f-divergences by Barthe and Olmedo [2].

Definition 5.1 ((f, δ)-Lifting of a relation Ψ)**.** Let $\Psi \subseteq T_1 \times T_2$, let f be a convex function providing an f-divergence and let $\delta \in \mathbb{R}^+$. Then, we have that $\mu_1 \in \mathfrak{M}[T_1]$ and $\mu_2 \in \mathfrak{M}[T_2]$ are in the (f, δ)-lifting of Ψ, denoted $\mathcal{L}_{(f,\delta)}(\Psi)$ iff there exist two distributions $\mu_L, \mu_R \in \mathfrak{M}[T_1 \times T_2]$ such that

1. $\mu_i\,(a, b) > 0$ implies $(a, b) \in \Psi$, for $i \in \{L, R\}$,
2. $\pi_1\,\mu_L = \mu_1 \wedge \pi_2\,\mu_R = \mu_2$, and
3. $\Delta_f(\mu_L, \mu_R) \leqslant \delta$.

where $\pi_1\,\mu = \lambda x. \sum_y \mu\,(x, y)$ and $\pi_2\,\mu = \lambda y. \sum_x \mu\,(x, y)$.

We will call the distributions μ_L and μ_R the left and right witnesses for the lifting, respectively.

This notion of lifting will be used to give a relational interpretation of monadic types. We say that a valuation θ validates a relational environment $\mathcal{G}$, denoted $\theta \Vdash \mathcal{G}$, if $\theta \vDash \|\mathcal{G}\|$ and $\forall x \in \mathrm{dom}(\mathcal{G})$, $(x_\lhd\theta, x_\rhd\theta) \in (\!|x\mathcal{G}|\!)_\theta$. The relational interpretation $[\![\phi]\!]_\theta \in \{\top, \bot\}$ of assertions ϕ with respect to a valuation $\theta \vDash \Gamma$ is an extension of the the one provided in Barthe et al. [6]. In Figure 7 we provide the extensions specific to PrivInfer. Notice that we interpret the assertion $\Delta_f^\mathfrak{D}(e_1^\bowtie, e_2^\bowtie) \leqslant \delta$ with the corresponding f-divergence.

$$\textsc{UnitM}\ \frac{\|\mathcal{G}\| \vdash f \in \mathcal{F} \qquad \|\mathcal{G}\| \vdash \delta : \overline{\mathbb{R}}^{+} \qquad \mathcal{G} \vdash e :: T}{\mathcal{G} \vdash \texttt{return}\ e :: \mathfrak{M}_{f,\delta}[T]} \qquad \textsc{Infer}\ \frac{\mathcal{G} \vdash e : \mathfrak{M}_{f,\delta}[\{x :: \widetilde{\tau} \mid x_{\triangleleft} = x_{\triangleright}\}]}{\mathcal{G} \vdash \mathsf{infer}(e) : \{x :: \mathfrak{D}[\widetilde{\tau}] \mid \Delta_f^{\mathfrak{D}}(x_{\triangleleft}, x_{\triangleright}) \leqslant \delta\}}$$

$$\textsc{BindM}\ \frac{\mathcal{G} \vdash e_1 :: \mathfrak{M}_{f_1,\delta_1}[T_1] \qquad (f_1, f_2)\ \text{are}\ f_3\text{-composable} \qquad \mathcal{G} \vdash \mathfrak{M}_{f_2,\delta_2}[T_2] \qquad \mathcal{G}, x :: T_1 \vdash e_2 :: \mathfrak{M}_{f_2,\delta_2}[T_2]}{\mathcal{G} \vdash \texttt{mlet}\ x = e_1\ \texttt{in}\ e_2 :: \mathfrak{M}_{f_3,\delta_1+\delta_2}[T_2]} \qquad \textsc{Ran}\ \frac{\mathcal{G} \vdash e : \{x :: \mathfrak{D}[\widetilde{\tau}] \mid \Delta_f^{\mathfrak{D}}(x_{\triangleleft}, x_{\triangleright}) \leqslant \delta\}}{\mathcal{G} \vdash \mathsf{ran}(e) : \mathfrak{M}_{f,\delta}[\{x :: \widetilde{\tau} \mid x_{\triangleleft} = x_{\triangleright}\}]}$$

$$\textsc{Observe}\ \frac{\begin{array}{c}\mathcal{G} \vdash e :: \mathfrak{M}_{f,\delta}[\{y :: \widetilde{\tau} \mid y_{\triangleleft} = y_{\triangleright}\}] \\ \mathcal{G}, x : \widetilde{\tau} \vdash e' :: \mathfrak{M}_{f,\delta'}[\{y :: \mathbb{B} \mid y_{\triangleleft} = y_{\triangleright}\}] \qquad \|\mathcal{G}\| \vdash \Delta_f(\mathsf{observe}\, x_{\triangleleft} \Rightarrow e_{\triangleleft}\, \mathsf{in}\, e'_{\triangleleft}, \mathsf{observe}\, x_{\triangleright} \Rightarrow e_{\triangleright}\, \mathsf{in}\, e'_{\triangleright}) \leqslant \delta''\end{array}}{\mathcal{G} \vdash \mathsf{observe}\, x \Rightarrow e\, \mathsf{in}\, e' :: \mathfrak{M}_{f,\delta''}[\{y :: \widetilde{\tau} \mid y_{\triangleleft} = y_{\triangleright}\}]}$$

Figure 5: PrivInfer Relational Typing Rules

$$\frac{d_1, d_2 \in [\![\widetilde{\tau}]\!]}{(d_1, d_2) \in (\!|\widetilde{\tau}|\!)_\theta} \qquad \frac{(d_1, d_2) \in (\!|T|\!)_\theta \qquad [\![\phi]\!]_{\theta\{x_{\triangleleft} \mapsto d_1\}\{x_{\triangleright} \mapsto d_2\}}}{(d_1, d_2) \in (\!|\{x :: T \mid \phi\}|\!)_\theta}$$

$$\frac{\begin{array}{c} f_1, f_2 \in [\![|T| \to |U|]\!] \\ \forall (d_1, d_2) \in (\!|T|\!)_\theta.\, (f_1(d_1), f_2(d_2)) \in (\!|U|\!)_{\theta\{x_{\triangleleft} \mapsto d_1\}\{x_{\triangleright} \mapsto d_2\}} \end{array}}{(f_1, f_2) \in (\!|\Pi(x :: T).\, U|\!)_\theta}$$

$$\frac{d_1, d_2 \in [\![\mathfrak{D}[\widetilde{\tau}]]\!]}{(d_1, d_2) \in (\!|\mathfrak{D}[\widetilde{\tau}]|\!)_\theta} \qquad \frac{\begin{array}{c}\mu_1, \mu_2 \in [\![\mathfrak{M}[|T|]]\!] \\ \mathcal{L}_{f,\delta}((\!|T|\!)_\theta)\ \mu_1\ \mu_2\end{array}}{(\mu_1, \mu_2) \in (\!|\mathfrak{M}_{f,\delta}[T]|\!)_\theta}$$

Figure 6: Relational interpretation of types

We give in Figure 6 the relational interpretation $(\!|T|\!)_\theta$ of a relational type T with respect to the valuation $\theta \models \|\mathcal{G}\|$. This corresponds to pairs of values in the standard interpretation of $\mathbf{PCF}_p$ expressions. To define this interpretation we use both the interpretation of relational assertions given in Figure 7 and the definition of lifting given in Definition 5.1. The interpretation of relational assertions is used in the interpretation of relational refinement types, while the lifting is used to provide interpretation to the probabilistic polymonad. Notice that the relational interpretation of a type $\mathfrak{D}[\widetilde{\tau}]$ is just the set of pairs of values in the standard interpretation of $\mathfrak{D}[\widetilde{\tau}]$. This can then be restricted by using relational refinement types. We can then prove that the relational refinement type system is sound with respect to the relational interpretation of types.

Theorem 5.1 (Soundness)**.** *If $\mathcal{G} \vdash e_1 \sim e_2 :: T$, then for every valuation $\theta \models \mathcal{G}$ we have $([\![e_1]\!]_\theta, [\![e_2]\!]_\theta) \in (\!|T|\!)_\theta$.*

The soundness theorem above give us a concrete way to reason about f-divergences.

Corollary 5.1 (f-divergence)**.** *If $\vdash e :: \mathfrak{M}_{f,\delta}[\{y :: \tau \mid y_{\triangleleft} = y_{\triangleright}\}]$ then for every $(\mu_1, \mu_2) \in [\![e]\!]$ we have $\Delta_f(\mu_1, \mu_2) \leqslant \delta$.*

Moreover, thanks to the characterization of differential privacy in terms of f-divergence given by Barthe and Olmedo [2] we can refine the previous result to show that PrivInfer accurately models differential privacy.

Corollary 5.2 (Differential Privacy)**.** *If $\vdash e :: \{x :: \sigma \mid \Phi\} \to \mathfrak{M}_{\epsilon\text{-D},\delta}\{y :: \tau \mid y_{\triangleleft} = y_{\triangleright}\}$ then $[\![e]\!]$ is (ϵ, δ)-differentially private w.r.t. adjacency relation $[\![\Phi]\!]$.*

$$[\![f \in \mathcal{F}]\!]_\theta = [\![f]\!]_\theta \in \mathcal{F}$$
$$[\![\Delta_f^{\mathfrak{D}}(e_1^{\bowtie}, e_2^{\bowtie}) \leqslant \delta]\!]_\theta = \Delta_{[\![f]\!]_\theta}([\![e_1^{\bowtie}]\!]_\theta, [\![e_2^{\bowtie}]\!]_\theta) \leqslant [\![\delta]\!]_\theta$$

Figure 7: Relational interpretation of assertions (added rules)

6. EXAMPLES

In this section we show how we can use PrivInfer to guarantee differential privacy for Bayesian learning by adding noise on the input, noise on the output using ℓ_1 norm, and noise on the output using f-divergences. We will show some of these approaches on three classical examples from Bayesian learning: learning the bias of a coin from some observations (as discussed in § 2), its generalized process, i.e the Dirichlet/multinomial model and the learning of the mean of a Gaussian. In all the example we will use pseudo code that can be easily desugared into the language presented in § 4. Indeed, the following examples have been type-checked with an actual tool implementing PrivInfer. More examples can be found in the supplementary material section.

6.1 Input perturbation

Input perturbation: Beta Learning.

Let's start by revisiting the task of inferring the parameter of a Bernoulli distributed random variable given a sequence of private observations. We consider two lists of booleans with the same length in the adjacency relation Φ iff they differ in the value of at most one entry. We want to ensure differential privacy by perturbing the input. A natural way to do this, since the observations are boolean value is by using the exponential mechanism. We can then learn the bias from the perturbed data. The post-processing property of differential privacy ensures that we will learn the parameter in a private way.

Let's start by considering the quality score function for the exponential mechanism. A natural choice is to consider a function `score:bool→bool→{0,1}` mapping equal booleans to 1 and different booleans to 0. Remember that the intended reading is that one of the boolean is the one to which we want to give a quality score, while the other is the one provided by the observation. The sensitivity of `score` is 1. Using this score function we can then create a general function for adding noise to the input list:

```
1.  let rec addNoise db eps  = match db with
2.  | []    → return ([])
3.  | y::yl → mlet yn = (expMech eps score y) in
4.            mlet yln = (addNoise yl eps) in return(yn::yln)
```

To this function we can give the following type guaranteeing differential privacy.

$$\{l :: \mathbb{B}\ \texttt{list} \mid l_{\triangleleft}\ \Phi\ l_{\triangleright}\} \to \{\epsilon :: \mathbb{R}^+ \mid =\} \to \mathfrak{M}_{\epsilon\text{-D},0}\{b :: \mathbb{B}\ \texttt{list} \mid =\}$$

where we use $\{x :: T \mid =\}$ as a shorthand for $\{x :: T \mid x_{\triangleleft} = x_{\triangleright}\}$. We will use this shorthand all along this section.

The bulk of the example is the following function that recursively updates the prior distribution and learns the final distribution over the parameter.

```
1.  let rec learnBias dbn prior = match dbn with
2.  | []     → prior
3.  | d::dbs → observe
4.         (fun r → mlet z = ran bernoulli(r) in return (d=z))
5.         (learnBias dbs prior)
```

The likelihood given in Line 4 is the formal version of the one we presented in § 2. The function learnBias can be typed in different ways depending on what is our goal. For this example we can assign to it the following type:

$$\{l :: \mathbb{B}\ \texttt{list} \mid =\} \to \mathfrak{M}_{\mathsf{SD},0}\{x :: [0,1] \mid =\} \to \mathfrak{M}_{\mathsf{SD},0}\{x :: [0,1] \mid =\}$$

The reading of this type is that if learnBias takes two lists of observations that are equal and two prior that are equal, then we obtain two posterior that are equal. Thanks to this we can type the occurrence of observe in line 3-4 using a trivial assertion. Here we use the SD divergence but in fact this would also hold for any other $f \in \mathcal{F}$. In particular, this type allows us to compose it with addNoise using an mlet. This type also reflects the fact that the prior is public. We can then compose these two procedures in the following program:

```
1.  let main db a b eps = mlet noisyDB = (addNoise db eps)
2.    in return(infer (learnBias noisyDB (ran (beta(a,b)))))
```

Notice that in line 2 we use infer for learning from the noised data. We can then assign to main the type

$$\{l ::\ \mathbb{B}\ \texttt{list} \mid l_{\triangleleft}\ \Phi\ l_{\triangleright}\} \to \{a :: \mathbb{R}^+ \mid =\} \to \{b :: \mathbb{R}^+\ \mid =\} \to \{\epsilon :: \mathbb{R}^+ \mid =\} \to \mathfrak{M}_{\epsilon\text{-D},0}\{d :: \mathfrak{D}[[0,1]] \mid =\}$$

which guarantees us that the result is ϵ differentially private. Notice that the result type is a polymonadic type over $\mathfrak{D}[[0,1]]$. This because we are releasing the symbolic distribution.

Input perturbation: Normal Learning.

An example similar to the previous one is learning the mean of a gaussian distribution with known variance: kv, from a list of real number observations—for instance some medical parameters like the level of LDL of each patient. We consider two lists of reals with the same length adjacent when the ℓ_1 distance between at most two elements (in the same position) is bounded by 1. To perturb the input we may now want to use a different mechanism, for example we could use the Gaussian mechanisms—this may give reasonable results if we expect the data to come from a normal distribution. Also in this case, the sensitivity is 1. The addNoise function is very similar to the one we used in the previous example:

```
1.  let rec addNoise db eps delta = match db with
2.  | []    → return ([])
3.  | y::yl → mlet yn = (gaussMech (sigma eps delta) y) in
4.          mlet yln = (addNoise yl eps delta) in return(yn::yln)
```

The two differences are that now we also have delta as input and that in line 3 instead of the score function we have a function sigma computing the variance as in Definition 3.4. The inference function become instead the following.

```
1.  let rec learnMean dbn prior = match dbn with
2.   | []     → prior
3.   | d::dbs → observe (fun (r: real) →
4.                mlet z = ran normal(r, kv) in return (d=z))
5.                (learnMean dbs prior) in
6.  let main db hMean hVar eps delta =
7.      mlet noisyDB = (addNoise db eps delta)in
8.        return(infer (learnMean noisyDB
9.                 (ran (normal(hMean,hVar)))))
```

Composing them we get the following type guaranteeing (ϵ, δ)-differential privacy.

$$\{l :: \mathbb{R}\ \texttt{list} \mid l_{\triangleleft}\ \Phi\ l_{\triangleright}\} \to \{a :: \mathbb{R}^+ \mid =\} \to \{b :: \mathbb{R}^+ \mid =\} \to \{\epsilon :: \mathbb{R}^+ \mid =\} \to \{\delta :: \mathbb{R}^+ \mid =\} \to \mathfrak{M}_{\epsilon\text{-D},\delta}\{d :: \mathfrak{D}[\mathbb{R}] \mid =\}$$

6.2 Noise on Output with ℓ_1-norm

We present examples where the privacy guarantee is achieved by adding noise on the output. For doing this we need to compute the sensitivity of the program. In contrast, in the previous section the sensitivity was evident because directly computed on the input. As discussed before we can compute the sensitivity with respect to different metrics. Here we consider the sensitivity computed over the ℓ_1-norm on the parameters of the posterior distribution.

Output parameters perturbation: Beta Learning.

The main difference with the example in the previous section is that here we add Laplacian noise to the parameters of the posterior.

```
1.  let main db a b eps=
2.   let  d = infer (learnBias db (ran beta(a,b))) in
3.    let (aP, bP) = getParams d in
4.    mlet aPn = lapMech(eps, aP) in
5.    mlet bPn = lapMech(eps, bP) in
6.    return beta(aPn, bPn)
```

In line 2 we use the function learnBias from the previous section, while in line 4 and 5 we add Laplace noise. The formal sensitivity analysis is based on the fact that the posterior parameters are going to be the counts of *true* and *false* in the data respectively summed up to the parameters of the prior. This reasoning is performed on each step of observe. Then we can prove that the ℓ_1-norm sensitivity of the whole program is 2 and type the program with a type guaranteeing 2ϵ-differentially privacy.

$$\{l :: \mathbb{B}\ \texttt{list} \mid l_{\triangleleft}\ \Phi\ l_{\triangleright}\} \to \{a :: \mathbb{R}^+ \mid =\} \to \{b :: \mathbb{R}^+ \mid =\} \to \{\epsilon :: \mathbb{R}^+ \mid =\} \to \mathfrak{M}_{2\epsilon\text{-D},0}\{d :: \mathfrak{D}[[0,1]] \mid =\}$$

Output parameters perturbation: Normal Learning.

For this example we use the same adjacency relation of the example with noise on the input where in particular the

number of observation n is public knowledge. The code is very similar to the previous one.

```
1. let main db hM hV kV eps =
2.   let mDistr = infer (learnMean db (ran normal(hM,kV))) in
3.   let  mean = getMean mDistr in
4.   mlet meanN = lapMech(eps/s mean) in
5.   let d = normal(meanN, uk) in return(d)
```

where $uk = \left(\frac{1}{hV^2} + \frac{n}{kv^2}\right)^{-1}$. Notice that we only add noise to the posterior mean parameter and not to the posterior variance parameter since the latter doesn't depend on the data but only on public information. The difficulty for verifying this example is in the sensitivity analysis. By some calculations this can be bound by $s = \frac{hV}{kv+hV}$ where kv is the known variance of the gaussian distribution whose mean we are learning and hV is the prior variance over the mean. We use this information in line 4 when we add noise with the Laplace mechanism. By using this information we can give the following type to the previous program:

$$\{l :: \mathbb{R}\ \texttt{list} \mid l_\triangleleft\ \Phi\ l_\triangleright\} \to \{hM :: \mathbb{R} \mid =\} \to \{hV :: \mathbb{R}^+ \mid =\} \to \{kv :: \mathbb{R}^+ \mid =\} \to \{\epsilon :: \mathbb{R}^+ \mid =\} \to \mathfrak{M}_{s\epsilon\text{-D},0}\{d :: \mathfrak{D}[\mathbb{R}] \mid =\}$$

6.3 Noise on Output using f-divergences

We now turn to the approach of calibrating the sensitivity according to f-divergences. We will consider once again the example for learning privately the distribution over the parameter of a Bernoulli distribution, but differently from the previous section we will add noise to the output of the inference algorithm using the exponential mechanism with a score function using an f-divergence. So, we perturb the output distribution and not its parameters.

We will use Hellinger distance as a metric over the output space of our differentially private program, but any other f-divergence could also be used. The quality score function for the exponential mechanism can be given a type of the shape:

$$\{l :: \mathbb{B}\ \texttt{list} \mid l_\triangleleft\ \Phi\ l_\triangleright\} \to \{d :: \mathfrak{D}[\tau] \mid =\} \to \{r :: \mathbb{R} \mid |r_\triangleleft - r_\triangleright| \leqslant \rho\}$$

where the bound ρ express its sensitivity. Now we can use as a score function the following program

```
score (db, prior) out = -(H (infer (learnBias db prior)) out)
```

This quality score uses a function H computing the Hellinger distance between the result of the inference and a potential output to assign it a score. The closer out is to the real distribution (using Hellinger distance), the higher the scoring is. If we use the exponential mechanism with this score we achieve our goal of using the Hellinger to "calibrate the noise". Indeed we have a program:

```
let main prior obs eps = expMech eps score (obs, prior)
```

To which we can assign type:

$$\mathfrak{M}_{\mathsf{HD},0}\{x :: [0,1] \mid =\} \to \{\ell :: \mathbb{B}\ \texttt{list} \mid \ell_\triangleleft\ \Phi\ \ell_\triangleright\} \to \{\epsilon :: \mathbb{R}^+ \mid =\} \to \mathfrak{M}_{\rho\epsilon\text{-D},0}\{d :: \mathfrak{D}[[0,1]] \mid =\}$$

Concretely, to achieve this we can proceed by considering first the code for learnBias:

```
1. let rec learnBias db prior = match dbn with
2. | []     → prior
3. | d::dbs → mlet rec = (learBias dbs prior) in observe
4.    (fun  r  →  mlet z = ran bernoulli(r) in return (d=z)) rec
```

To have a bound for the whole learnBias we need first to give a bound to the difference in Hellinger distance that two distinct observations can generate. This is described by the following lemma.

Lemma 6.1. *Let $d_1, d_2 : \mathbb{B}$ with $d_1 \Phi d_2$. Let $a, b \in \mathbb{R}^+$. Let $\Pr(\xi) = \mathsf{Beta}(a,b)$. Then $\Delta_{\mathsf{HD}}(\Pr(\xi \mid d_1), \Pr(\xi \mid d_2)) \leqslant \sqrt{1 - \frac{\pi}{4}} = \rho$.*

Using this lemma we can then type the observe statement with the bound ρ. We still need to propagate this bound to the whole learnBias. We can do this by using the adjacency relation which imposes at most one difference in the observations, and the data processing inequality Theorem 3.1 guaranteeing that for equal observations the Hellinger distance cannot increase. Summing up, using the lemma above, the adjacency assumption and the data processing inequality we can give to learnBias the following type:

$$\{l :: \mathbb{B}\ \texttt{list} \mid l_\triangleleft\ \Phi\ l_\triangleright\} \to \mathfrak{M}_{\mathsf{HD},0}\{x :: [0,1] \mid =\} \to \mathfrak{M}_{\mathsf{HD},\rho}\{x :: [0,1] \mid =\}$$

This ensures that starting from the same prior and observing l_1 and l_2 in the two different runs such that $l_1\ \Phi\ l_2$ we can achieve two beta distributions which are at distance at most ρ. Using some additional refinement for infer and H we can guarantee that score has the intended type, and so we can guarantee that overall this program is $(\rho\epsilon, 0)$-differential privacy.

The reasoning above is not limited to the Hellinger distance. For instance the following lemma:

Lemma 6.2. *Let $d_1, d_2 : \mathbb{B}$ with $d_1 \Phi d_2$. Let $a, b \in \mathbb{R}^+$. Let $\Pr(\xi) = \mathsf{Beta}(a,b)$. Then $\Delta_{\mathsf{SD}}(\Pr(\xi \mid d_1), \Pr(\xi \mid d_2)) \leqslant \sqrt{2(1 - \frac{\pi}{4})} = \zeta$.*

gives a type in term of statistical distance:

$$\{l :: \mathbb{B}\ \texttt{list} \mid l_\triangleleft\ \Phi\ l_\triangleright\} \to \mathfrak{M}_{\mathsf{SD},0}\{x :: [0,1] \mid =\} \to \mathfrak{M}_{\mathsf{SD},\zeta}\{x :: [0,1] \mid =\}$$

The choice of which metric to use is ultimately left to the user.

This example easily extends also to the Dirichlet example. More details about the Dirichlet distribution are given in the supplementary material section. Indeed, Lemma 6.1 can be generalized to arbitrary Dirichlet distributions:

Lemma 6.3. *Let $k \in \mathbb{N}^{\geqslant 2}$, $d_1, d_2 : [k]\ \texttt{list}$ with $d_1 \Phi d_2$. Let $a_1, a_2, \ldots, a_k \in \mathbb{R}^+$. Let $\Pr(\xi) = \mathsf{Dirichlet}(a_1, a_2, \ldots, a_k)$. Then $\Delta_{\mathsf{HD}}(\Pr(\xi \mid d_1), \Pr(\xi \mid d_2)) \leqslant \sqrt{1 - \frac{\pi}{4}} = \rho$.*

Using this lemma we can assign to the following program:

```
1. let rec learnP db prior = match dbn with
2. | []     → prior
3. | d::dbs → mlet rec = (learnP dbs prior) in observe
4.    (fun  r  s→  mlet z = ran multinomial(r,s) in
5.                                  return (d=z)) rec
```

the type:

$$\{l :: [\mathbf{3}]\ \texttt{list} \mid l_\triangleleft\ \Phi\ l_\triangleright\} \to \mathfrak{M}_{\mathsf{HD},0}\{x :: [0,1]^2 \mid =\} \to \mathfrak{M}_{\mathsf{HD},\rho}\{x :: [0,1]^2 \mid =\}$$

Similarly to the previous example we can now add noise to the output of the inference process using the sensitivity with respect to the Hellinger distance and obtain a $(\rho\epsilon, 0)$-differential privacy guarantee.

7. RELATED WORK AND CONTRIBUTIONS

Differential privacy and Bayesian inference. Our system targets programs from the combination of differential privacy and Bayesian inference. Both of these topics are active areas of research, and their intersection is an especially popular research direction today. We briefly summarize the most well-known work, and refer interested readers to surveys for a more detailed development (Dwork and Roth [16] for differential privacy, Bishop [9] for Bayesian inference).

Blum et al. [10] and Dwork et al. [17] proposed *differential privacy*, a worst-case notion of statistical privacy, in a pair of groundbreaking papers, initiating intense research interest in developing differentially private algorithms. The original works propose the Laplace and Gaussian mechanisms that we use, while the seminal paper of McSherry and Talwar [30] introduces the exponential mechanism. Recently, researchers have investigated how to guarantee differential privacy when performing *Bayesian inference*, a foundational technique in machine learning. Roughly speaking, works in the literature have explored three different approaches to guaranteeing differential privacy when the samples are private data. First, we may add noise directly to the samples, and then perform inference as usual [37]. Second, we may perform inference on the private data, then add noise to the parameters themselves [39]. This approach requires bounding the sensitivity of the output parameters when we change a single data sample, relying on specific properties of the model and the prior distribution. The final approach involves no noise during inference, but outputs samples from the posterior rather than the entire posterior distribution [14, 39, 40]. This last approach is highly specific to the model and prior, and our system does not handle this method of achieving privacy, yet.

Formal verification for differential privacy. In parallel with the development of private algorithms, researchers in formal verification have proposed a wide variety of techniques for *verifying* differential privacy. For a comprehensive discussion, interested readers can consult the recent survey by Barthe et al. [8]. Many of these techniques rely on the composition properties of privacy, though there are some exceptions [7]. For a brief survey, the first systems were based on runtime verification of privacy [29]. The first systems for static verification of privacy used linear type systems [21, 33]. There is also extensive work on relational program logics for differential privacy [2–4], and techniques for verifying privacy in standard Hoare logic using product programs [5]. None of these techniques have been applied to verifying differential privacy of Bayesian inference. Our system is most closely related to HOARe^2, a relational refinement type system that was recently proposed by Barthe et al. [6]. This system has been used for verifying differential privacy of algorithms, and more general relational properties like incentive compatibility from the field of mechanism design. We extend this approach in three directions:

- we provide a relational typing rule for observe (conditional distributions) and for infer (Bayesian inference),
- we provide a generalization of the relational type system of HOARe^2 to reason about symbolic and actual distributions,
- we generalize the probability polymonad of HOARe^2 to reason about general f-divergence.

The combination of these three contributions allows us to address Bayesian inference, which is not supported by HOARe^2.

Probabilistic programming. Research in probabilistic programming has emerged early in the 60s and 70s, and is nowadays a very active research area. Relevant to our work is in particular the research in probabilistic programming for machine learning and statistics which has been very active in recent years. Many probabilistic programming languages have been designed for these applications, including WinBUGS [28], IBAL [32], Church [22], Infer.net [31], Tabular [24], Anglican [35], Dr. Bayes [36]. Our goal is not to provide a new language but instead is to propose a framework where one can reason about differential privacy for such languages. For instance, we compiled programs written in Tabular [24] into PrivInfer so that differential privacy could be verified. More information on this translation can be found in the supplementary material section. Another related work is the one by Adams and Jacobs [1] proposing a type theory for Bayesian inference. While technically their work is very different from our it shares the same goal of providing reasoning principles for Bayesian inference.

8. CONCLUSION

We have presented PrivInfer, a type-based framework for differentially private Bayesian inference. Our framework allows to write data analysis as functional programs for Bayesian inference and add to noise to them in different ways using different metrics. Besides, our framework allows to reason about general f-divergences for Bayesian inference.

Future directions include exploring the use of this approach to guarantee robustness for Bayesian inference and other machine learning techniques [15], to ensure differential privacy using conditions over the prior and the likelihood similar to the ones studied by Zhang et al. [39], Zheng [40], and investigating further uses of f-divergences for improving the utility of differentially private Bayesian learning. On the programming language side it would also be interesting to extend our framework to continuous distributions following the approach by Sato [34]. We believe that the intersection of programming languages, machine learning, and differential privacy will reserve us many exciting results.

References

[1] R. Adams and B. Jacobs. A type theory for probabilistic and bayesian reasoning. *CoRR*, abs/1511.09230, 2015.

[2] G. Barthe and F. Olmedo. Beyond differential privacy: Composition theorems and relational logic for f-divergences between probabilistic programs. In *ICALP*, 2013.

[3] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella-Béguelin. Probabilistic Relational Reasoning for Differential Privacy. In *POPL*, 2012.

[4] G. Barthe, G. Danezis, B. Grégoire, C. Kunz, and S. Zanella Béguelin. Verified computational differential privacy with applications to smart metering. In *CSF*, 2013.

[5] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, C. Kunz, and P.-Y. Strub. Proving differential privacy in Hoare logic. In *CSF*, 2014.

[6] G. Barthe, M. Gaboardi, E. J. G. Arias, J. Hsu, A. Roth, and P. Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. In *POPL*, 2015.

[7] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P.-Y. Strub. Proving differential privacy via probabilistic couplings. In *LICS*, 2016.

[8] G. Barthe, M. Gaboardi, J. Hsu, and B. Pierce. Programming language techniques for differential privacy. *ACM SIGLOG News*, 2016.

[9] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2006. ISBN 0387310738.

[10] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: The SuLQ framework. In *PODS*, 2005.

[11] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. 2011.

[12] I. Csiszár. Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizitat von Markoffschen Ketten. *Magyar. Tud. Akad. Mat. Kutató Int. Közl*, 1963.

[13] I. Csiszár and P. Shields. Information theory and statistics: A tutorial. *Foundations and Trends in Communications and Information Theory*, 2004.

[14] C. Dimitrakakis, B. Nelson, A. Mitrokotsa, and B. I. P. Rubinstein. Robust and Private Bayesian Inference. In *ALT*, 2014.

[15] L. R. B. Dipak K. Dey. Robust Bayesian analysis using divergence measures. *Statistics & Probability Letters*, 1994.

[16] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 2014.

[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[18] C. Dwork, G. N. Rothblum, and S. P. Vadhan. Boosting and differential privacy. In *FOCS*, 2010.

[19] H. Ebadi, D. Sands, and G. Schneider. Differential privacy: Now it's getting personal. *POPL*, 2015.

[20] F. Eigner and M. Maffei. Differential privacy by typing in security protocols. In *CSF*, 2013.

[21] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *POPL*, 2013.

[22] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *UAI*, 2008.

[23] A. D. Gordon, M. Aizatulin, J. Borgström, G. Claret, T. Graepel, A. V. Nori, S. K. Rajamani, and C. V. Russo. A model-learner pattern for bayesian reasoning. In *POPL*, 2013.

[24] A. D. Gordon, T. Graepel, N. Rolland, C. V. Russo, J. Borgström, and J. Guiver. Tabular: a schema-driven probabilistic programming language. In *POPL*, 2014.

[25] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, 2012.

[26] M. Hicks, G. M. Bierman, N. Guts, D. Leijen, and N. Swamy. Polymonadic programming. In *MSFP*, 2014.

[27] S. Katsumata. Parametric effect monads and semantics of effect systems. In *POPL*, 2014.

[28] D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter. WinBUGS - A bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing*, 2000.

[29] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *International Conference on Management of Data*, 2009.

[30] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, 2007.

[31] T. Minka, J. Winn, J. Guiver, and D. Knowles. Infer.NET 2.5, 2012. URL http://research.microsoft.com/infernet. MSR.

[32] A. Pfeffer. IBAL: A Probabilistic Rational Programming Language. In *IJCAI*, 2001.

[33] J. Reed and B. C. Pierce. Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy. In *ICFP*, 2010.

[34] T. Sato. Approximate Relational Hoare Logic for Continuous Random Samplings. *CoRR*, abs/1603.01445.

[35] D. Tolpin, J. van de Meent, and F. Wood. Probabilistic Programming in Anglican. In *ECML PKDD*, 2015.

[36] N. Toronto, J. McCarthy, and D. V. Horn. Running Probabilistic Programs Backwards. In *ESOP*, 2015.

[37] O. Williams and F. McSherry. Probabilistic Inference and Differential Privacy. In *NIPS*, 2010.

[38] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private data release via bayesian networks. In *SIGMOD*, 2014.

[39] Z. Zhang, B. I. P. Rubinstein, and C. Dimitrakakis. On the Differential Privacy of Bayesian Inference. In *AAAI*, 2016.

[40] S. Zheng. The differential privacy of Bayesian inference, 2015. URL http://nrs.harvard.edu/urn-3:HUL.InstRepos:14398533. Bachelor's thesis, Harvard College.