

Probabilistic Data with Continuous Distributions*

Martin Grohe¹, Benjamin Lucien Kaminski², Joost-Pieter Katoen¹,
and Peter Lindner¹

¹RWTH Aachen University, Germany

²University College London, UK

Statistical models of real world data typically involve continuous probability distributions such as normal, Laplace, or exponential distributions. Such distributions are supported by many probabilistic modelling formalisms, including probabilistic database systems. Yet, the traditional theoretical framework of probabilistic databases focuses entirely on finite probabilistic databases.

Only recently, we set out to develop the mathematical theory of infinite probabilistic databases. The present paper is an exposition of two recent papers which are cornerstones of this theory. In (Grohe, Lindner; ICDT 2020) we propose a very general framework for probabilistic databases, possibly involving continuous probability distributions, and show that queries have a well-defined semantics in this framework. In (Grohe, Kaminski, Katoen, Lindner; PODS 2020) we extend the declarative probabilistic programming language Generative Datalog, proposed by (Bárány et al. 2017) for discrete probability distributions, to continuous probability distributions and show that such programs yield generative models of continuous probabilistic databases.

1 Introduction

Probabilistic databases [20, 21, 22] provide a framework for quantitatively modelling uncertainty in data. Sources of uncertainty are numerous; common examples are noisy

*The results presented in this paper were originally published in M. Grohe and P. Lindner: *Infinite Probabilistic Databases*, Proc. ICDT 2020 and M. Grohe, B.L. Kaminski, J.-P. Katoen, P. Lindner: *Generative Datalog with Continuous Distributions*, Proc. PODS 2020.

RoomNo	Time	°C
4108	2021-01-05 08:00	20.2
4108	2021-01-05 14:00	21.8
4109	2021-01-05 08:00	22.1
⋮	⋮	⋮

(a)

RoomNo	Time	°C
4108	2021-01-05 08:00	$\mathcal{N}(20.2, \varepsilon)$
4108	2021-01-05 14:00	$\mathcal{N}(21.8, \varepsilon)$
4109	2021-01-05 08:00	$\mathcal{N}(22.1, \varepsilon)$
⋮	⋮	⋮

(b)

Figure 1: A relation storing room temperatures: (a) deterministically and (b) with a normally distributed error

sensor data, data gathered from unreliable sources, and inconsistent data. Formally, a probabilistic database (PDB) is a probability space over database instances, called the possible worlds. Traditionally, these probability spaces were limited to be finite. This implies a closed world assumption where only finitely many facts could possibly be true, and it rules out any probability distributions with an infinite support. Yet, in many applications, infinitely, even uncountably infinitely, supported probability distributions arise naturally, and many real-world statistical phenomena follow infinite probability distributions such as Poisson distributions, normal distributions, or exponential distributions.

Example 1.1. Suppose we have a relation storing room temperatures, as in Figure 1(a). The temperature measurements may be noisy, and we may account for this by adding a normally distributed error with a small variance $\varepsilon > 0$, resulting in a simple probabilistic relation which may be represented as in Figure 1(b). ┘

Example 1.2. In this example, consider a particle detector such as the Alpha Magnetic Spectrometer (AMS-02¹) on the ISS. Suppose we record the detected particles in a relation of schema $(\text{Time}, \text{Trajectory}, \text{Velocity})$. As in the previous example, the measurements (of the trajectory and velocity) may be imprecise and best modelled by a probability distribution. But here we have an additional source of uncertainty: some particles may go undetected. If we also model this type of error, the number of tuples in the relation becomes a random variable as well. Then there is no a-priori bound on the size of the instances in the resulting PDB. Note, however, that every instance is still finite, because in every time interval only finitely many particles can hit the detector, and our model should account for that. ┘

Both examples exhibit probabilistic databases with continuous probability distributions that cannot be captured by the traditional model of finite probabilistic databases. Generalising from finite to continuous probability distributions comes with a substantial mathematical overhead. While PDBs of fixed (or bounded) size, such as those arising from Example 1.1, are still relatively easy to handle, PDBs of unbounded size such as the one we saw in Example 1.2 are nontrivial to capture mathematically, let alone to

¹See <https://ams02.space/>.

deal with algorithmically. Several PDB systems that have been proposed over the years [2, 13, 14, 19] handle continuous probability distributions. The flexibility of these systems reaches as far as providing declarative representations of continuous probabilistic databases and even continuous-space database-valued Markov processes. Yet, only recently [10, 12], we proposed a general framework for rigorously dealing with probabilistic databases over continuous domains and provided a sound semantics for standard query languages such as the relational calculus. We will present this framework in Sections 4 and 5 of this paper. To distinguish them from the traditional “finite” PDBs, we call PDBs with an infinite sample space *infinite PDBs* in the following. Note that every instance in an infinite PDB is just a standard finite relational database instance, it is only the sample space of all possible instances that is infinite.

A difficult issue when dealing with PDBs is how to efficiently represent them. This problem already arises for finite PDBs, but is much more fundamental when dealing with infinite PDBs that do not even allow for a naive representation that explicitly lists all instances. So we have to rely on *implicit* representations, which can either be ad-hoc representations such as the one chosen to illustrate Example 1.1 in Figure 1(b) or generic formalisms for representing complex probability distributions, such as probabilistic graphical models, deep neural networks, and probabilistic programming languages. Yet, when dealing with (relational) PDBs, it is desirable to stay within the declarative framework of relational databases. To this end, Bárány, ten Cate, Kimelfeld, Olteanu, and Vagena [3] introduced a declarative probabilistic programming language based on Datalog, which has a generative part enabling to represent complex probability distributions strictly within the framework of relational databases. However, the semantics of Bárány et al. is only able to handle discrete probability distributions. In [9], we generalised the semantics to continuous distributions. The resulting *Generative Datalog* can serve both as a powerful representation language for relational PDBs with discrete and continuous distributions and as a query language for PDBs. We present this language in Section 7.

The reader may wonder if it is really necessary to consider continuous probabilistic databases. After all, they can only be mathematical abstractions of real systems, where instead of the continuum of real numbers we only see the finite set of 64 bit floating point numbers. Then aren’t finite probabilistic databases all we need? Well, the history of computer science has shown us that the right abstractions can be extremely powerful—just think of the relational database model—and certainly we do not want the semantics of our query languages depend on whether we use 32 or 64 bit floating-point numbers to specify probabilities. All of applied mathematics, including statistics, uses the real numbers as the right abstraction to reason about continuous phenomena. And when reasoning about uncertain and probabilistic data, we want to have standard tools such as normal distributions at our disposal.

2 Towards Infinite PDBs

Before we delve into the mathematical details, in this section we describe the general approach on an intuitive level and highlight the technical difficulties we are facing.

We define a probabilistic database to be a probability space whose sample space consists of database instances of some schema τ . In the traditional approach, this probability space is assumed to be finite; here, we would like to allow it to be infinite. The difficulty when defining probabilities on uncountable spaces such as the reals is that we cannot assign a well-defined probability to all subsets of the space, but only to subsets that are *measurable*.

Let us ignore this issue for a moment (though it will come back to bite us) and think about *how we can actually define a probability distribution on uncountable sets of database instances*. Let us fix a simple database schema τ consisting of a single binary relation R of schema $(\text{Time}, \text{Value})$, where the attribute Value is real-valued. Instances are relations of this schema. We can also view them as finite sets (without duplicates) or finite bags (possibly with duplicates)—depending on the type of semantics we are interested in—of facts of the form $R(t, v)$, where t is a point in time and v a real number. If we want to define a finite probability space on the instances, we can simply pick a finite set D_1, \dots, D_m of instances and assign probabilities $p_1, \dots, p_m \in [0, 1]$ to them such that $\sum_i p_i = 1$. We can extend this approach to countably infinite spaces, but not to uncountable spaces, where typically every single instance has probability 0. This happens, for example, if we assume the Value to be normally distributed at any Time . We know how to define a probability distribution on the Values (that is, the real numbers); we only need to specify the probability mass on each interval. But here we need to define a probability distribution on *sets* or *bags* of Time-Value pairs. It is not at all obvious how to do that, except maybe in simple settings such as the one described in Example 1.1. We need to draw from the theory of *finite point processes* [17, 16, 7]. In probability theory, point processes are used to describe probability spaces of finite or countable sets or bags. Based on the theory of point processes, we will define a very general framework for infinite PDBs that we call *standard PDBs* (see Section 4).

Once we have defined our probability spaces, we need to think about *querying PDBs*. To define the semantics of queries and views, let us consider a view V mapping instances of schema τ to instances of schema τ' . Queries are just specific views where the target schema τ' consist of a single relation schema. We want to define a semantics for this view V on probabilistic databases, that is, we want to extend it to a mapping from PDBs of schema τ to PDBs of schema τ' . Let us assume that we have a PDB \mathcal{D} of schema τ , and we want to define the image $V(\mathcal{D})$, which is supposed to be a PDB of schema τ' . To do this, for a set \mathbf{D}' of instances of schema τ' we define the probability of \mathbf{D}' in $V(\mathcal{D})$ to be the probability of the set $V^{-1}(\mathbf{D}')$ in \mathcal{D} :

$$\Pr_{V(\mathcal{D})}(\mathbf{D}') := \Pr_{\mathcal{D}}(V^{-1}(\mathbf{D}')). \quad (\text{A})$$

Example 2.1. Recall Example 1.1, where we considered PDBs of a schema

$$\tau = \{\text{Temp}(\text{RoomNo}, \text{Time}, \text{°C})\}.$$

RoomNo	Time	°C
4108	2021-01-05 08:00	$\mathcal{N}(20.2, 0.1)$
4108	2021-01-05 14:00	$\mathcal{N}(21.8, 0.1)$
4109	2021-01-05 08:00	$\mathcal{N}(22.1, 0.1)$
4109	2021-01-05 14:00	$\mathcal{N}(22.3, 0.1)$
4109	2021-01-06 08:00	$\mathcal{N}(21.9, 0.1)$

Figure 2: A PDB of schema $\tau = \{\text{Temp}(\text{RoomNo}, \text{Time}, \text{°C})\}$

Entries are room temperatures at various times. Let Q be the query that maps instances of schema τ to instances of schema $\tau' = \{\text{AvTemp}(\text{RoomNo}, \text{°C})\}$ recording the average temperature in each room, defined by the SQL-expression

SELECT RoomNo, AVG(°C) FROM Temp GROUP BY RoomNo.

Let us apply this query to the PDB \mathcal{D} represented by the relation **Temp** shown in Figure 2. Note that in all instances of this PDB, the table **Temp** has exactly five rows recording the temperatures in room 4108 at two different times and the temperatures in room 4109 at three different times. For simplicity, we assume that the random variables describing the entries in the five rows are independent.

In every instance of $Q(\mathcal{D})$, the table **AvTemp** has exactly two rows recording the average temperatures in rooms 4108 and 4109. We can easily compute the probabilities in $Q(\mathcal{D})$. For example, the probability that both rooms have an average temperature in the range 20–22 degrees equals the probability that the average of two normally distributed random variables with means 20.2, 21.8 and variance 0.1 is between 20 and 22 times the probability that the average of three normally distributed random variables with means 22.1, 22.3, 21.9 and variance 0.1 is between 20 and 22. Actually, the table **AvTemp** in $Q(\mathcal{D})$ can be represented as follows.

RoomNo	°C
4108	$\mathcal{N}(21.0, 0.05)$
4109	$\mathcal{N}(22.1, 0.033)$

The fact that a linear combination of normal distributions is again a normal distribution enables us to represent $Q(\mathcal{D})$ in such a simple “closed form”. In general, views of PDBs can be far more complicated than the original PDBs. \lrcorner

Unfortunately, there is a subtle issue that we have neglected when defining the semantics of views and queries over PDBs. Recall that in uncountable probability spaces, we cannot define probabilities for all subsets of the sample space, but only for so-called *measurable* sets. This means that in the definition (A), we only need to consider measurable sets \mathbf{D}' of instances of schema τ' , but we need to make sure that the set $V^{-1}(\mathbf{D}')$ is measurable as well, for otherwise the probability on the right-hand side of (A) is not defined. This means that a view V only has a well-defined semantics on probabilistic databases if for every measurable set \mathbf{D}' in the target space the pre-image $V^{-1}(\mathbf{D}')$ is

a measurable set in the source space. If this is the case, we call V measurable. *Only measurable views and queries have a well-defined semantics on probabilistic databases.* Fortunately, it turns out that all views defined in standard query languages such as the relational calculus or Datalog are measurable. But this is a nontrivial result (Theorem 5.1). In [12, Example 8], we give an example of a relatively simple “query” that is not measurable.

3 Mathematical Background

In this section, we collect some mathematical background underlying our approach to PDBs. The reader may skip this section and use it as a reference whenever needed later.

Topology

Topology qualitatively captures concepts such as closeness, convergence, and continuity, and it is the foundation for the measure theory and probability theory we need here. A *topology* on a set \mathbb{X} is a family \mathfrak{D} of subsets of \mathbb{X} that contains \mathbb{X} and the empty set and is closed under arbitrary unions and finite intersections. We call $(\mathbb{X}, \mathfrak{D})$ a *topological space* and the elements $O \in \mathfrak{D}$ *open sets*.

Example 3.1. (1) In the *standard topology* on the reals \mathbb{R} , a set $O \subseteq \mathbb{R}$ is open if for every $x \in O$ there is an $\varepsilon > 0$ such that $(x - \varepsilon, x + \varepsilon) \subseteq O$. Note that this topology is *generated* by the open intervals, which means that every open set is the union of open intervals.²

(2) For every set \mathbb{X} , the power set $2^{\mathbb{X}}$ is a topology on \mathbb{X} , the so-called *discrete topology*.
 \lrcorner

For $i = 1, 2$, let $(\mathbb{X}_i, \mathfrak{D}_i)$ be a topological space. A function $f: \mathbb{X}_1 \rightarrow \mathbb{X}_2$ is *continuous* (with respect to $\mathfrak{D}_1, \mathfrak{D}_2$) if $f^{-1}(O_2) \in \mathfrak{D}_1$ for every $O_2 \in \mathfrak{D}_2$.

Every *metric* d on \mathbb{X} (that is, a distance function on pairs of elements of \mathbb{X} that is symmetric, satisfies the triangle inequality, and has the property that two points have distance 0 if and only if they are equal) induces a topology on \mathbb{X} where a set $O \subseteq \mathbb{X}$ is open if for every $x \in O$ there is an $\varepsilon > 0$ such that $\{y \mid d(x, y) < \varepsilon\} \subseteq O$. A topological space $(\mathbb{X}, \mathfrak{D})$ is *metrisable* if it is induced by a metric on \mathbb{X} in this way. Obviously, the standard topology on the reals (Example 3.1(1)) is metrisable. The discrete topology on an arbitrary set \mathbb{X} (Example 3.1(2)) is metrisable as well; as a metric we use the function d with $d(x, x) = 0$ and $d(x, y) = 1$ for all $x \neq y$, also known as the *discrete metric*.

A topological space $(\mathbb{X}, \mathfrak{D})$ is *separable* if there is a countable subset $Y \subseteq \mathbb{X}$ such that every nonempty open set $O \in \mathfrak{D} \setminus \{\emptyset\}$ contains an element from Y (we say that Y is *dense*). For example, for the reals with the standard topology, the set \mathbb{Q} of rationals is a dense subset. The discrete topology on a set \mathbb{X} is separable if and only if \mathbb{X} is countable. Separability is a very important technical property in our arguments, because it enables us to work with countable approximations.

²We take the union over the empty family of sets to be the empty set.

A final condition we need (though it is less important for us) is completeness: intuitively, a metrisable topological space $(\mathbb{X}, \mathfrak{D})$ is *complete* if every convergent sequence (more precisely, Cauchy sequence) converges to a point in \mathbb{X} . We omit the formal definition. A *Polish space* is a complete, separable, metrisable topological space (and its topology is *Polish*). The reals with the standard topology, all finite-dimensional Euclidean spaces, and all countable discrete topological spaces are Polish spaces.

It is safe to say that all topological spaces we will ever find in database applications are Polish spaces.

Measure Theory and Probability

A σ -algebra on a set \mathbb{X} is a set \mathfrak{A} of subsets of \mathbb{X} that contains the empty set and is closed under complementation and countable unions. A pair $(\mathbb{X}, \mathfrak{A})$, where \mathfrak{A} is a σ -algebra on \mathbb{X} , is called a *measurable space*.

- Example 3.2.** (1) For every set \mathbb{X} , the set $\{\emptyset, \mathbb{X}\}$ and the power set $2^{\mathbb{X}}$ are σ -algebras on \mathbb{X} .
(2) Another σ -algebra on \mathbb{X} is the set of all $Y \subseteq \mathbb{X}$ such that either Y is countable or $\mathbb{X} \setminus Y$ is countable.
(3) The set of all Lebesgue measurable subsets of the reals is a σ -algebra. ┘

Let \mathbb{X} be a set and $\mathfrak{S} \subseteq 2^{\mathbb{X}}$. The σ -algebra *generated* by \mathfrak{S} is the closure of \mathfrak{S} under complementation and countable intersections, that is, the smallest σ -algebra on \mathbb{X} that contains \mathfrak{S} . Observe that the σ -algebra defined in Example 3.2(2) is the σ -algebra generated by all singleton sets $\{x\}$ for $x \in \mathbb{X}$.

For any topological space (X, \mathfrak{D}) , the σ -algebra generated by the topology \mathfrak{D} is called the *Borel σ -algebra* on \mathbb{X} , and its elements are called *Borel sets*. A measurable space $(\mathbb{X}, \mathfrak{A})$ is a *standard Borel space* if \mathfrak{A} is the Borel σ -algebra of some Polish topology on \mathbb{X} . It is not difficult to show that if d is a metric inducing such a Polish topology and Y is a countable dense subset then \mathfrak{A} is generated by the countable set of open balls $B_{1/n}(y) := \{x \in \mathbb{X} \mid d(x, y) < 1/n\}$ for positive integers n and $y \in Y$. This is one of the reasons making standard Borel spaces very convenient to handle.

For $i = 1, 2$, let $(\mathbb{X}_i, \mathfrak{A}_i)$ be a measurable space. A function $f: \mathbb{X}_1 \rightarrow \mathbb{X}_2$ is *measurable* (with respect to $\mathfrak{A}_1, \mathfrak{A}_2$) if $f^{-1}(A_2) \in \mathfrak{A}_1$ for every $A_2 \in \mathfrak{A}_2$. If \mathfrak{A}_i is the Borel σ -algebra of some topology \mathfrak{D}_i on \mathbb{X}_i , then every continuous function is measurable; the converse does not always hold. The *Cartesian product* of $(\mathbb{X}_1, \mathfrak{A}_1)$ and $(\mathbb{X}_2, \mathfrak{A}_2)$ is the measurable space $(X_1 \times X_2, \mathfrak{A}_1 \otimes \mathfrak{A}_2)$, where $\mathfrak{A}_1 \otimes \mathfrak{A}_2$ is the σ -algebra generated by the sets $A_1 \times A_2$ for $A_i \in \mathfrak{A}_i$. If \mathbb{X}_1 and \mathbb{X}_2 are disjoint, then the (disjoint) union of the two measurable spaces is the measurable space $(X_1 \cup X_2, \mathfrak{A}_1 \oplus \mathfrak{A}_2)$, where $\mathfrak{A}_1 \oplus \mathfrak{A}_2$ is the set of all sets $A \subseteq X_1 \cup X_2$ such that $A \cap X_i \in \mathfrak{A}_i$. It can be shown that if the spaces $(\mathbb{X}_i, \mathfrak{A}_i)$ are standard Borel spaces then $(X_1 \times X_2, \mathfrak{A}_1 \otimes \mathfrak{A}_2)$ and $(X_1 \cup X_2, \mathfrak{A}_1 \oplus \mathfrak{A}_2)$ are standard Borel spaces as well.

Let $(\mathbb{X}, \mathfrak{A})$ be a measurable space. A *measure* on $(\mathbb{X}, \mathfrak{A})$ is a function M from \mathfrak{A} to $\mathbb{R}_{\geq 0} \cup \{\infty\}$ (the nonnegative reals extended by infinity) that is σ -additive, that is, for every countable family A_1, A_2, \dots of mutually disjoint sets in \mathfrak{A} it holds that $M(\bigcup_{i \geq 1} A_i) =$

$\sum_{i \geq 1} M(A_i)$. A measure M is *finite* if $M(\mathbb{X}) < \infty$, and it is a *probability measure* (or a *probability distribution*) if $M(\mathbb{X}) = 1$. We call $(\mathbb{X}, \mathfrak{A}, M)$ a *measure space*, or a *probability space* if M is a probability measure. \mathbb{X} is called the *sample space* and \mathfrak{A} the *event space* of this probability space.

Example 3.3. (1) Let $(\mathbb{X}, \mathfrak{A})$ be a measurable space and $Y \subseteq \mathbb{X}$. We define a measure M by letting $M(A)$ be the cardinality of $|A \cap Y|$ (either finite or ∞). M is what we call a *counting measure*. It is finite if Y is finite.

(2) The *normal distribution* $\mathcal{N}(\mu, \sigma)$ is the unique probability measure P on the standard Borel space $(\mathbb{R}, \mathfrak{B})$ with

$$P((a, b]) = \frac{1}{\sigma\sqrt{2\pi}} \int_a^b \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) dx.$$

for all $a < b$. (It can be shown that a probability measure on the Borel σ -algebra over the reals is determined by its values on the half open intervals.)

(3) If \mathbb{X} is a countable set (finite or countably infinite), then we can define a probability measure P on $(\mathbb{X}, 2^{\mathbb{X}})$ by defining the singleton values $P(\{x\}) \in [0, 1]$ such that $\sum_{x \in \mathbb{X}} P(\{x\}) = 1$ and letting $P(A) := \sum_{x \in A} P(\{x\})$ for all $A \subseteq \mathbb{X}$. In fact, every probability measure on \mathbb{X} can be defined this way, regardless of what the σ -algebra is. So for countable probability spaces, we can always assume that the σ -algebra is the power set of the sample space \mathbb{X} . \lrcorner

4 Standard PDBs

Let τ be a database schema, and let \mathbb{U}_τ , the *universe*, be the union of the domains of all attributes occurring in τ . We view *database instances* as finite sets or bags (a.k.a. multisets) of *facts* of the form $R(a_1, \dots, a_k)$, where $R(A_1, \dots, A_k)$ is a relation schema in τ and for every i the value $a_i \in \mathbb{U}_\tau$ is contained in the domain of attributes A_i . Even if we are only interested in set instances, for technical reasons we need to consider bag instances as well. We denote the set of all facts over τ by \mathbb{F}_τ and the set of database instances over τ , that is, finite bags of facts in \mathbb{F}_τ , by $\mathbb{DB}_\tau^{\text{bag}}$. Moreover, we denote the subset of all plain sets of facts, that is, set instances, by \mathbb{DB}_τ . In all these notations, we omit the subscript τ if the schema is clear from the context or irrelevant.

Under the most general definition, a *probabilistic database* is just a probability space $\mathcal{D} = (\mathbb{D}, \mathfrak{A}, P)$, where $\mathbb{D} \subseteq \mathbb{DB}_\tau^{\text{bag}}$. However, it is very difficult to work with this general definition. While we may have an intuition about defining the sample space and the probabilities, it is completely unclear how to define the event space, that is, the σ -algebra \mathfrak{A} , which is a set of sets of bags of facts (sic).

Example 4.1. Recall Examples 1.1, 2.1, and let $\mathcal{D} = (\mathbb{D}, \mathfrak{A}, P)$ be the (informally

described) PDB shown in Figure 2. The sample space $\mathbb{D} \subseteq \mathbb{DB}_\tau$ consists of all instances

$$\begin{aligned} &\{ \text{Temp}(4108, 2021-01-05 \ 08:00, t_1), \\ &\quad \text{Temp}(4108, 2021-01-05 \ 14:00, t_2), \\ &\quad \text{Temp}(4109, 2021-01-05 \ 08:00, t_3), \\ &\quad \text{Temp}(4109, 2021-01-05 \ 14:00, t_4), \\ &\quad \text{Temp}(4109, 2021-01-06 \ 08:00, t_5) \}, \end{aligned}$$

where $t_1, \dots, t_5 \in \mathbb{R}$. We have seen in Example 2.1 how to calculate the probability of a set of instances. However, it is not obvious which sets are measurable, that is, have a well-defined probability and therefore should belong to the σ -algebra \mathfrak{A} . Intuitively, at least sets such as those considered in Example 2.1 where the temperatures are in certain intervals, should be measurable.

In this simple example, we can define a suitable σ -algebra by an ad-hoc product construction starting from the Borel σ -algebra on the reals, but already in the only slightly more complicated setting of Example 1.2, where the number of tuples in an instance is also a random variable that is a-priori unbounded, it becomes difficult to carry out such a construction. \lrcorner

The point is: even if we can somehow come up with an ad-hoc construction of a σ -algebra for every PDB that we want to work with, reasoning about σ -algebras is definitely not what we want to do when working with probabilistic data. Yet, as we have seen in Section 2, measurability is an issue when giving queries and views a meaningful semantics.

A solution to this dilemma is a theoretical framework that gives us a *generic* construction of σ -algebras only depending on the schema τ and the universe \mathbb{U} that is rich enough to make all sets that we typically want to consider measurable and at the same time ensures that all reasonable queries and views are measurable. *Standard probabilistic databases*, introduced in [12], provide such a framework.

The main technical challenge is to generically construct a sufficiently rich σ -algebra $\mathfrak{A} = \mathfrak{A}_\tau$ on \mathbb{DB}^{bag} . If the universe \mathbb{U} is countable, then the set \mathbb{F} of facts and hence the set \mathbb{DB}^{bag} of all finite bags of facts are countable as well, and we can simply let $\mathfrak{A} = 2^{\mathbb{DB}^{\text{bag}}}$ (see Example 3.3(3)). But what do we do if the universe is uncountable? The additional assumption we need to make is that we have a *topology on the universe*, in fact a *Polish topology*. As uncountable universes we may see in typical database applications are usually derived from the reals in some way, this is no serious restriction.

Example 4.2. Typical domains of database attributes are integers, reals, strings, and time stamps. So we will have a universe like $\mathbb{U} = \Sigma^* \cup \mathbb{R}$ for some finite alphabet Σ (say, UTF8). If by $\mathfrak{D}_{\mathbb{R}}$ we denote the standard topology on the reals, the generic way of extending it to a Polish topology \mathfrak{D} on \mathbb{U} is to let \mathfrak{D} be the set of all $O \subseteq \mathbb{U}$ such that $O \cap \mathbb{R} \in \mathfrak{D}_{\mathbb{R}}$. It is straightforward to extend this construction to more complicated universes where we add, for example, a set of (uncountably many) time stamps. \lrcorner

Let us assume in the following that $\mathfrak{D}_{\mathbb{U}}$ is a Polish topology on the universe \mathbb{U} . We assume that this topology is part of the information provided by the schema τ . Let $\mathfrak{A}_{\mathbb{U}}$ be the σ -algebra generated by $\mathfrak{D}_{\mathbb{U}}$. Then $(\mathbb{U}, \mathfrak{A}_{\mathbb{U}})$ is a standard Borel space. Using finite Cartesian products and disjoint unions, we can lift $\mathfrak{A}_{\mathbb{U}}$ to a σ -algebra $\mathfrak{A}_{\mathbb{F}}$ on the set \mathbb{F} of facts. $(\mathbb{F}, \mathfrak{A}_{\mathbb{F}})$ is still a standard Borel space.

The next step will be to lift $\mathfrak{A}_{\mathbb{F}}$ to a σ -algebra \mathfrak{C} on \mathbb{DB}^{bag} . Maybe the most direct way of doing this is to first lift the σ -algebra to all finite tuples of facts using finite Cartesian products and a countable disjoint union and then “factor” the resulting σ -algebra through all permutations to go from tuples to bags. A more elegant way of defining the same σ -algebra is as follows.³ For every set $F \subseteq \mathbb{F}$ of facts and every instance $D \in \mathbb{DB}^{\text{bag}}$, we let $|D|_F$ be the number of elements of F in D counted according to their multiplicities. For example, $|\{\{f, f, g, g, g, h\}\}_{\{f, g\}}| = 5$. For $n \in \mathbb{N}$, we let $\#(F, n)$ be the set of all $D \in \mathbb{DB}^{\text{bag}}$ with $|D|_F = n$. Finally, we let $\mathfrak{C} := \mathfrak{C}_{\tau}$ be the σ -algebra generated by all sets $\#(F, n)$ for $F \in \mathfrak{A}_{\mathbb{F}}$ and $n \in \mathbb{N}$. Since \mathfrak{C} is generated by the *counting events* $\#(F, n)$, it is called the *counting σ -algebra* (hence the letter \mathfrak{C} = “Fraktur C”). Another way of seeing \mathfrak{C} is that it is the smallest σ -algebra such that for all measurable sets $F \in \mathfrak{A}_{\mathbb{F}}$ of facts, the function $|\cdot|_F: \mathbb{DB}^{\text{bag}} \rightarrow \mathbb{N}$ is measurable with respect to \mathfrak{C} and $2^{\mathbb{N}}$. We will see that this is enough to guarantee that all queries defined in standard query languages are measurable as well.

Definition 4.3. A *standard probabilistic database* is a probability space $(\mathbb{DB}_{\tau}^{\text{bag}}, \mathfrak{C}_{\tau}, P)$ for some schema τ .

To keep the definition as simple as possible, we let the sample space of a standard PDB be the set $\mathbb{DB}_{\tau}^{\text{bag}}$ of all bag instances. As a result, every standard PDB can be specified by its probability distribution. We could adopt a more liberal definition where the sample space consists of an arbitrary measurable subset and then restrict the σ -algebra to this set. That is, we could also admit PDBs of the form $(\mathbb{D}, \mathfrak{C}|_{\mathbb{D}}, P)$, where $\mathbb{D} \in \mathfrak{C}$ and $\mathfrak{C}|_{\mathbb{D}} := \{C \cap \mathbb{D} \mid C \in \mathfrak{C}\}$. But note that this space is essentially the same as the standard PDB $(\mathbb{DB}^{\text{bag}}, \mathfrak{C}, P')$ where $P'(C) := P(C \cap \mathbb{D})$ for all $C \in \mathfrak{C}$. Therefore, it is safe to view such PDBs with restricted sample spaces as standard PDBs. In particular, since the set \mathbb{DB} of all set instances is measurable, this applies to *standard set PDBs* of the form $(\mathbb{DB}, \mathfrak{C}|_{\mathbb{DB}}, P)$.

5 Query Semantics

A *view* with *input schema* τ and *output schema* τ' is a mapping $V: \mathbb{DB}_{\tau}^{\text{bag}} \rightarrow \mathbb{DB}_{\tau'}^{\text{bag}}$. A *query* is a view where the output schema consists of a single relation. We call a view $V: \mathbb{DB}_{\tau}^{\text{bag}} \rightarrow \mathbb{DB}_{\tau'}^{\text{bag}}$ *measurable* if it is a measurable mapping with respect to \mathfrak{C}_{τ} and $\mathfrak{C}_{\tau'}$. Such a measurable view V can be lifted to standard PDBs as follows: for every standard PDB $\mathcal{D} = (\mathbb{DB}_{\tau}^{\text{bag}}, \mathfrak{C}_{\tau}, P)$, let $V(\mathcal{D})$ be the standard PDB $(\mathbb{DB}_{\tau'}^{\text{bag}}, \mathfrak{C}_{\tau'}, P')$ where P' is

³It is not obvious that the two constructions indeed lead to the same σ -algebra. This follows from a theorem from point-process theory.

defined by

$$P'(C') := P(V^{-1}(C))$$

for all $C' \in \mathfrak{C}_{\tau'}$. Note that this is exactly semantics defined in (A).

Thus a view has a well-defined semantics on standard PDBs if and only if it is measurable. The following theorem, which is the main result of [12], states that this is the case for a wide class of views.

Theorem 5.1. *All queries and views expressible in the relational calculus (with aggregation) and Datalog as well as variants such as Inflationary Datalog and Least Fixed-Point Logic (see [1]) are measurable.*

Let us remark that this theorem applies to both set semantics and bag semantics. The proof is a tedious inductive proof that goes through all operators used to define the different query languages. The most involved steps are basic relational-algebra operators such as Cartesian product or projection. The following example exhibits some of the arguments in an easy case that nevertheless already illustrates why we want our underlying topological space to be Polish.

Example 5.2. Let $\tau = \{R(A, B)\}$, where the attributes A, B have the same domain \mathbb{U} . We consider the equality query Q that maps R to its diagonal, that is, the selection

$$\text{SELECT } A, B \text{ FROM } R \text{ WHERE } A = B.$$

Let d be a metric on \mathbb{U} that induces the Polish topology $\mathfrak{D}_{\mathbb{U}}$ we assume to exist, and let $Y \subseteq \mathbb{U}$ be a countable dense set. Let $\mathfrak{A}_{\mathbb{U}}$ be the Borel σ -algebra on \mathbb{U} . Then $\mathfrak{A}_{\mathbb{U}}$ is generated by the open balls $B_{1/n}(y) = \{x \in \mathbb{U} \mid d(x, y) < 1/n\}$ for $y \in Y$ and $n \in \mathbb{N}_{>0}$. Let $\mathfrak{A}_{\mathbb{F}}$ be the lifted σ -algebra on \mathbb{F} , and $\Delta := \{R(x, y) \in \mathbb{F} \mid x = y\}$ be the diagonal selected by the query Q .

As a first step, we need to prove that $\Delta \in \mathfrak{A}_{\mathbb{F}}$. This is done by characterising its complement Δ^c . We identify the space $\mathbb{F} = \{R(x, y) \mid x, y \in \mathbb{U}\}$ with the Cartesian product $\mathbb{U} \times \mathbb{U}$ and $\mathfrak{A}_{\mathbb{F}}$ with the product σ -algebra $\mathfrak{A}_{\mathbb{U}} \otimes \mathfrak{A}_{\mathbb{U}}$ generated by the sets $A \times A'$ for $A, A' \in \mathfrak{A}_{\mathbb{U}}$. Then, Δ becomes $\{(x, x) \mid x \in \mathbb{U}\}$. Observe that for $(x, x') \in \Delta^c$, there are $y, y' \in Y$ and an $n \in \mathbb{N}_{>0}$ such that $(x, x') \in B_{1/n}(y) \times B_{1/n}(y')$ and $(B_{1/n}(y) \times B_{1/n}(y')) \cap \Delta = \emptyset$. Thus

$$\Delta^c = \bigcup_{\substack{y, y' \in Y \\ B_{1/n}(y) \times B_{1/n}(y') \cap \Delta = \emptyset}} B_{1/n}(y) \times B_{1/n}(y'),$$

which is a countable union of sets in $\mathfrak{A}_{\mathbb{U}} \otimes \mathfrak{A}_{\mathbb{U}}$. Since every σ -algebra is closed under complementation and countable intersections, it follows that $\Delta \in \mathfrak{A}_{\mathbb{U}} \otimes \mathfrak{A}_{\mathbb{U}}$.

To prove that the query Q , formally a mapping from $\mathbb{DB}_{\tau}^{\text{bag}}$ to $\mathbb{DB}_{\tau}^{\text{bag}}$, is measurable, we need to prove that for every $C \in \mathfrak{C}$ the pre-image $Q^{-1}(C)$ is in \mathfrak{C} as well. As the counting events $\#(F, n)$ for $n \in \mathbb{N}$ and $F \in \mathfrak{A}_{\mathbb{F}}$ generate \mathfrak{C} , it suffices to prove that the pre-image of each such counting event is in \mathfrak{C} . Observe that for every instance $D \in \mathbb{DB}^{\text{bag}}$

we have $Q(D) \in \#(F, n)$ if and only if D contains exactly n facts $R(x, y) \in F$ with $x = y$ (counted according to multiplicity), or equivalently, $R(x, y) \in F \cap \Delta$. That is,

$$Q^{-1}(\#(F, n)) = \#(F \cap \Delta, n).$$

$F, \Delta \in \mathfrak{A}_{\mathbb{F}}$ imply $F \cap \Delta \in \mathfrak{A}_{\mathbb{F}}$ and thus $\#(F \cap \Delta, n) \in \mathfrak{C}$. ┘

6 Representations

An infinite PDB viewed as a probability distribution over database instances is an idealised mathematical concept that allows us to give semantics to PDBs and queries. It is not something that we can ever materialise. When designing probabilistic database systems, we need to think about finite representations of the probability spaces.

The most common model for finite probabilistic databases is that of *tuple-independent (TI)* PDBs. We can adopt the notion of TI PDBs to countable PDBs; to represent a countably infinite TI PDB we only need to represent a function that assigns a probability to every fact. Countably infinite TI PDBs were studied in [10]. An extension to uncountable PDBs, called *Poisson PDBs*, was proposed in [11]. Another basic model, *block-independent disjoint (BID)* PDBs, can also be extended to the infinite setting [10, 11]. Both TI and BID PDBs can only represent very simple probability distributions. To obtain more sophisticated distributions, we can apply transformations such as views to such PDBs (views of countable TI and BID PDBs were studied in [6]) or combine several PDBs into a new one using constructions such as convex combinations and superpositions (see [11]).

A different and more general approach is to start from a deterministic set of data, feed it into some generative model, and interpret the output as a probability distribution on database instances. With this approach, we are free to use all kinds of probabilistic modelling formalisms, for example, database-valued Markov processes [13], logical formalisms such as Markov Logic Networks [18] or ProbLog [8], deep neural network models such as variational autoencoders [15], or programs in some probabilistic programming language [4]. The difficulty is to specify such models in a way that the output can be interpreted as a meaningful probability distribution on database instances. Generative Datalog (GDatalog), introduced in [3, 9], is a declarative probabilistic programming language that remains within the framework of relational databases and therefore avoids this difficulty; the output of a Generative Datalog program is a PDB by definition.

7 Generative Datalog

Throughout this section, we assume a set semantics for relational databases. For recursive languages like Datalog, a bag semantics is less convenient, because we want to avoid repeatedly generating new copies of the same fact.

We start by informally reviewing Datalog (see [1] for more background). A *Datalog program* is a finite set of *rules* of the form

$$R(\bar{x}) \leftarrow S_1(\bar{x}_1), \dots, S_m(\bar{x}_m), \tag{B}$$

where R, S_i are relation symbols and \bar{x}, \bar{x}_i are tuples of variables of the appropriate lengths such that all variables in the tuple \bar{x} appear in one of the tuples \bar{x}_i . The *head* of the rule (B) is $R(\bar{x})$ and the *body* is $S_1(\bar{x}_1), \dots, S_m(\bar{x}_m)$. The relations appearing in the head of some rule of a Datalog program are *intensional*; all other relations are *extensional*. The extensional (intensional) relations form the *extensional (intensional, resp.) schema* of the program.

Consider the rule (B). Given an interpretation of all the body relations S_i and an assignment α to the variables \bar{x}_i , the rule is applicable if for all i the fact $S_i(\alpha(\bar{x}_i))$ holds true under the current interpretation of S_i . The application of the rule generates the new fact $R(\alpha(\bar{x}))$.

We run a Datalog program \mathcal{P} on a database instance over the extensional schema. The program iteratively computes interpretations for all the intensional relation symbols. All intensional relations are initialised to be empty. Then the rules of the program are applied repeatedly until no more new facts can be generated. It can be shown that the final relations do not depend on the order in which the rules are applied. Furthermore, the program (when applied to a finite input instance) always terminates in finitely many steps and hence the result can be interpreted as a database instance over the intensional schema. In other words, a Datalog program expresses a view mapping instances over the extensional schema to instances over the intensional schema.

The following example illustrates these Datalog definitions and then develops the main ideas of its probabilistic extension *GDatalog*.

Example 7.1. Let \mathcal{P} be the following simple Datalog program with extensional relations S, E and an intensional relation R :

$$R(x) \leftarrow S(x), \tag{C}$$

$$R(x) \leftarrow R(y), E(y, x). \tag{D}$$

We can interpret instances over the extensional schema as directed graphs with a distinguished set of source vertices. Then the program computes the set of all vertices reachable from the source vertices.

Now suppose we do not only want to compute whether a vertex is reachable from a source, but also how long it takes to reach it. Assume that the edge relation is now ternary, where we interpret the third, real valued component as a length or traversal time. Consider the following program:

$$R(x, 0) \leftarrow S(x), \tag{E}$$

$$R(x, t + s) \leftarrow R(y, t), E(y, x, s). \tag{F}$$

This is no longer a Datalog program in the strict sense. Yet it seems clear what its semantics is; after executing the program, the binary relation R will contain all pairs (x, t) such that x is reachable from a source vertex by a walk (that is, a path with possibly repeated vertices and edges) of length t . There is however a problem with this: if the graph is cyclic, there may be arbitrarily long walks, and the output relation will no longer be finite. Therefore, let us assume that the input graph is acyclic.

Now assume the traversal time of an edge is not deterministic, but random. Say, we model it by a log-normal distribution $\mathcal{LN}(\ln s, 1/10)$ where the parameter s is the median of this distribution. We may write the following program:

$$R(x, 0) \leftarrow S(x), \tag{G}$$

$$R(x, t + \mathcal{LN}(\ln s, 1/10)) \leftarrow R(y, t), E(y, x, s). \tag{H}$$

The output of this program is supposed to be a random relation that contains pairs (x, t) , where t is a sampled travel time along some walk from a source to x in the input graph. We can interpret the probability space of all possible output relations as a PDB of schema $\{R\}$. Our *GDatalog program* applied to an acyclic input graph thus represents a PDB.

But now another problem with termination pops up, even if the input graph is acyclic. The intuitive interpretation of an application of rule (H) is that for x, t, s, y matching the body of the rule, we sample a value r from the log-normal distribution $\mathcal{LN}(\ln s, 1/10)$ and then generate the fact $R(x, t+r)$. But if we would apply the same rule again to the same x, t, s , almost surely we would not sample the same r again, but an $r' \neq r$, and hence generate a new fact $R(x, t+r')$. We could do this over and over again and would obtain an infinite relation R , and moreover, the program would never terminate. This is clearly not what we want. Note that this cannot happen with a deterministic rule like (F).

Our simple solution to avoid this problem is to stipulate that *we can only apply the rule once to every triple (x, t, s) of parameters*. There may, however, be application scenarios where it is desirable to sample from the distribution more than once. To accommodate this, *we allow the same rule to appear several times in a program*. Then for each instantiation of the rule, we can sample once. A more flexible, but more complicated way of sampling several times with the same parameter tuple is to introduce another parameter that serves as an index for the samples. \lrcorner

Rules (G) and (H) give a typical example of a GDatalog program. To define GDatalog programs in general, besides the extensional and intensional schema we need to specify a family Ψ of *parameterised distributions*. An example is the log-normal distribution $\mathcal{LN}(\mu, \sigma)$ with parameters the real number μ and the positive real number σ . It may be helpful to think of parameterised distributions as randomised functions mapping the parameters, such as μ and σ , to values in some range, in the case of $\mathcal{LN}(\mu, \sigma)$ the positive reals. As a second example, consider the simple Bernoulli (coin-flip) distribution $\mathcal{B}(p)$ with parameter $p \in (0, 1)$. It takes value 1 with probability p and 0 with probability $1-p$. The functions in Ψ must satisfy some technical measurability conditions to ensure that they behave well with respect to changes of parameters. Intuitively, we want continuous changes in the parameters to result in continuous changes of the distribution, whatever that means technically. As examples, think of a normal distribution $\mathcal{N}(\mu, \sigma)$ and a Bernoulli distribution $\mathcal{B}(p)$. We can compose the parameterised distributions and replace parameters by constants to form more complex terms, but we need to make sure that the resulting parameterised distributions still satisfy our technical conditions. We call these Ψ -*terms*. An example of such a term, with two variables s, t , is the expression

$t + \mathcal{LN}(\ln s, 1/10)$ in rule (H). Deterministic functions such as $t + s$ can be easily incorporated into the semantics as well. A *GDatalog rule* over a set Ψ of parametrised distributions is an expression

$$R(\bar{t}) \leftarrow S_1(\bar{x}_1), \dots, S_m(\bar{x}_m), \quad (\text{I})$$

where the *body* $S_1(\bar{x}_1), \dots, S_m(\bar{x}_m)$ is a list of atoms over the extensional and intensional schema, just like for normal Datalog rules, and the *head* $R(\bar{t})$ consists of an intensional relation symbol R and a tuple $\bar{t} = (t_1, \dots, t_k)$ of (Ψ -)terms such that all variables of the t_i appear in the body of the rule. Of course we must make sure that the terms are of the appropriate types, that is, the range of t_i is contained in the domain of the i th attribute of relation R . Note that in particular, all normal Datalog rules are GDatalog rules.

A *GDatalog program* is a bag of GDatalog rules.

Before even touching upon the intricacies of a formal semantics for GDatalog programs, let us explain an informal operational semantics for GDatalog rules and programs. We have already given the intuition in Example 7.1. Consider the rule (I). Let $\bar{t} = (t_1, \dots, t_k)$, and let \bar{y}_i be the tuple of variables of the term t_i —we indicate this by writing $t_i(\bar{y}_i)$. Note that for Ψ -terms, $t_i(\bar{y}_i)$ is a parametrised probability distribution: if we instantiate the variables in \bar{y}_i by values of the appropriate type, we obtain a probability distribution.

Given an interpretation of all the body relations S_i and an assignment α to the variables \bar{x}_i , the rule is applicable if for all i the fact $S_i(\alpha(\bar{x}_i))$ holds true under the current interpretation of S_i . To apply the rule, for all j we sample a value a_j from the probability distributions $t_j(\alpha(\bar{y}_j))$ to generate the new fact $R(a_1, \dots, a_k)$.

We run a GDatalog program \mathcal{G} on a database instance over the extensional schema in a similar way as a normal Datalog program. All intensional relations are initialised to be empty. By repeatedly applying the rules as described above, the program generates (random) facts. All rule applications are stochastically independent. We stipulate that each rule of the program (or more precisely, each occurrence of each rule—remember a program is a bag of rules where a rule may occur several times) can only be applied once for every instantiation of the variables appearing in the head of the rule. The computation terminates if no more rule can be applied, and the output is the set of facts generated by the program during the computation, that is, a database instance over the intensional schema. Because of the sampling of values in the rule applications, the output is probabilistic. We interpret it as a probabilistic database. Thus, given a database instance over the extensional schema, a GDatalog program generates a PDB over the intensional schema.

However, our informal description of the semantics raises several crucial questions:

- (1) Does the program always terminate?
- (2) How can we be sure that the output is indeed a well-defined probabilistic database?
- (3) In which order do we apply the rules, and does this make a difference?

The answer to Question (1) is simply 'no' (in general). In Example 7.1, we already saw that even the simple deterministic program (E), (F) may not terminate if the input graph is cyclic. For probabilistic programs, the notion of termination is more complicated,

because a program may terminate for certain random outcomes while it diverges for other outcomes [5]. We resolve this issue by conditioning the output probability distribution on termination. That is to say, a GDatalog program defines a *sub-probabilistic database* where the probability mass over its defined space may be smaller than 1. It is an open research question to understand termination criteria for GDatalog programs.

We can answer Questions (2) and (3) by carefully defining a formal semantics for GDatalog programs. The main results of [9] (in the case of discrete probability distributions due to [3]) regarding this semantics are (informally) summarised in the following theorem.

Theorem 7.2. *Applying the GDatalog program \mathcal{G} to a database instance D over the extensional schema, defines a standard sub-probabilistic database $\mathcal{G}(D)$.*

$\mathcal{G}(D)$ does not depend on the order in which the rules are applied, as long as the policy that is used to decide which rules to apply is measurable.

One final remark is that the semantics of GDatalog programs can be lifted to probabilistic databases. That is, if we apply a GDatalog program to a standard PDB over the extensional schema it defines a standard sub-probabilistic database over the extensional schema.

8 Concluding Remarks

To enable reasoning about uncertain data with standard statistical models, we need probabilistic databases to support continuous probability distributions. Providing a mathematical framework for dealing with a very general class of probability distributions, we introduced standard PDBs [12]. Furthermore, we extended Generative Datalog [3], a declarative probabilistic programming language for relational data, to continuous distributions, thereby providing a flexible formalism for specifying generative models of standard probabilistic databases.

The focus of our work was on semantical issues. Further research is needed to address algorithmic and complexity theoretic questions.

Acknowledgements. This research is supported by the German Research Foundation (DFG) under grant GR 1492/16-1 and the Research Training Group 2236 UnRAVeL.

References

- [1] S. Abiteboul, R. Hull, and R. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] P. Agrawal and J. Widom. Continuous Uncertainty in Trio. In *Proc. VLDB Workshop on Management of Uncertain Data*, pages 17–32, 2009.
- [3] V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena. Declarative Probabilistic Programming with Datalog. *ACM Transactions on Database Systems (TODS)*, 42(4), 2017.

- [4] G. Barthe, J.-P. Katoen, and A. Silva, editors. *Foundations of Probabilistic Programming*. Cambridge University Press, 2020.
- [5] O. Bournez and F. Garnier. Proving positive almost-sure termination. In *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2005.
- [6] N. Carmeli, M. Grohe, P. Lindner, and C. Standke. Tuple-independent representations of infinite probabilistic databases. *ArXiv*, 2008.09511, 2020.
- [7] D. J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes, Volume I: Elementary Theory and Models*. Probability and Its Applications. Springer, 2nd edition, 2003.
- [8] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *Proc. IJCAI 2007*, pages 2468–2473.
- [9] M. Grohe, B. L. Kaminski, J.-P. Katoen, and P. Lindner. Generative datalog with continuous distributions. In *Proc. PODS 2020*, pages 347–360, New York, NY, USA.
- [10] M. Grohe and P. Lindner. Probabilistic Databases with an Infinite Open-World Assumption. In *Proc. PODS 2019*, pages 17–31.
- [11] M. Grohe and P. Lindner. Independence in infinite probabilistic databases. *ArXiv*, 2011.00096, 2020.
- [12] M. Grohe and P. Lindner. Infinite Probabilistic Databases. In *Proc. ICDT 2020*, pages 16:1–16:20, 2020.
- [13] R. Jampani, F. Xu, M. Wu, L. Perez, C. Jermaine, and P. J. Haas. The Monte Carlo Database System: Stochastic Analysis Close to the Data. *ACM Transactions on Database Systems (TODS)*, 36(3):18:1–18:41, 2011.
- [14] O. Kennedy and C. Koch. PIP: A Database System for Great and Small Expectations. In *Proc. ICDE 2010*, pages 157–168.
- [15] D. P. Kingma and M. Welling. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.
- [16] O. Macchi. The Coincidence Approach to Stochastic Point Processes. *Advances in Applied Probability*, 7(1):83–122, 1975.
- [17] J. E. Moyal. The General Theory of Stochastic Population Processes. *Acta Mathematica*, 108:1–31, 1962.
- [18] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1–2):107–136, 2006.

- [19] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database Support for Probabilistic Attributes and Tuples. In *Proc. ICDE 2008*, pages 1053–1061.
- [20] D. Suciu. Probabilistic databases for all. In *Proc. PODS 2020*, pages 19–31, 2020.
- [21] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool, 2011.
- [22] G. Van den Broeck and D. Suciu. Query Processing on Probabilistic Data: A Survey. *Foundations and Trends[®] in Databases*, 7(3–4):197–341, 2017.