Open access • Proceedings Article • DOI:10.1145/3375395.3387659

Generative Datalog with Continuous Distributions — Source link

Martin Grohe, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Peter Lindner

Institutions: RWTH Aachen University, University College London

Published on: 14 Jun 2020 - Symposium on Principles of Database Systems

Topics: Probabilistic programming language, Datalog, Probabilistic database, Probabilistic logic and Semantics (computer science)

Generative Datalog with Continuous Distributions

Martin Grohe
RWTH Aachen University
Aachen, Germany
grohe@informatik.rwth-aachen.de

Benjamin Lucien Kaminski
RWTH Aachen University
Aachen, Germany
University College London
London, United Kingdom
b.kaminski@ucl.ac.uk

Joost-Pieter Katoen
RWTH Aachen University
Aachen, Germany
katoen@informatik.rwth-aachen.de

Peter Lindner
RWTH Aachen University
Aachen, Germany
lindner@informatik.rwth-aachen.de

ABSTRACT

Arguing for the need to combine declarative and probabilistic programming, Bárány et al. (TODS 2017) recently introduced a probabilistic extension of Datalog as a "purely declarative probabilistic programming language." We revisit this language and propose a more foundational approach towards defining its semantics. It is based on standard notions from probability theory known as stochastic kernels and Markov processes. This allows us to extend the semantics to continuous probability distributions, thereby settling an open problem posed by Bárány et al.

We show that our semantics is fairly robust, allowing both parallel execution and arbitrary chase orders when evaluating a program. We cast our semantics in the framework of infinite probabilistic databases (Grohe and Lindner, ICDT 2020), and we show that the semantics remains meaningful even when the input of a probabilistic Datalog program is an arbitrary probabilistic database.

CCS CONCEPTS

• **Mathematics of computing** → *Probabilistic representations*; • **Theory of computation** → *Constraint and logic programming*; *Database query languages (principles)*; *Incomplete, inconsistent, and uncertain databases.*

KEYWORDS

Datalog; Probabilistic Databases; Generative Datalog; Measure Theory; Stochastic Kernels; Probabilistic Programming

ACM Reference Format:
Martin Grohe, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Peter Lindner. 2020. Generative Datalog with Continuous Distributions. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3375395.3387659

PODS'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-7108-7/20/06…$15.00
https://doi.org/10.1145/3375395.3387659

1 INTRODUCTION

Augmenting programming languages with stochastic behavior such as probabilistic choices or random sampling has a long tradition in computer science. In recent years, a lot of effort went into the development of dedicated probabilistic programming languages (see e.g. Anglican [39], Church [17], Figaro [33], Pyro [4], R2 [32], Stan [7]) that allow the specification and "execution", via probabilistic inference, of sophisticated probabilistic models. Such languages are nowadays important tools in a large variety of applications in different fields like artificial intelligence, computer vision, and cryptography to name a few [16, 18, 40].

From a database perspective, it is desirable to have a declarative probabilistic programming language that operates on a standard relational data model. Bárány, ten Cate, Kimelfeld, Olteanu, and Vagena [3] have recently introduced *Probabilistic Programming Datalog (PPDL)* which is relational and declarative in nature, while employing main features of probabilistic programming languages. PPDL, however, comes with the restriction that it only allows sampling from *discrete* probability distributions. As an open question, Bárány et al. [3] ask for an extension of their programming language semantics to continuous probability distributions. In this paper, we provide such an extension.

Probabilistic databases (PDBs) are a formal model for describing uncertainty in data [1, 38]. Traditionally, they were limited to probability spaces that consist of a finite number of possible alternative database instances (called *possible worlds*). Continuous probability distributions, however, arise naturally in many application scenarios that involve uncertain data like noisy sensor measurements. Moreover, for example, a lot of real world statistical phenomena, especially those that concern aspects of human behavior, follow normal or lognormal distributions [30].

Unfortunately, generalizing from discrete to continuous distributions usually comes with substantial mathematical overhead. While several systems [2, 24, 35] handle continuous probability distributions, only recently [21, 22], Grohe and Lindner proposed a general framework for rigorously dealing with probabilistic databases over continuous domains. Moreover, they establish basic properties such as the measurability of relational calculus and Datalog queries, which in turn allows for formally specifying the semantics of queries over continuous probabilistic databases. This framework

and some of the basic results, specifically the measurability of relational calculus queries, is also the foundation for this work.

The main contribution of this paper is a formal semantics for an extension of the probabilistic Datalog of Bárány et al. [3] allowing for sampling from continuous probability distributions. We focus on the "generative" component of probabilistic Datalog (called *Generative Datalog (GDatalog)* in [3]). A second component augmenting probabilistic Datalog by constraints is not considered in this paper. GDatalog programs generate a probabilistic database (that is, a probability distribution on database instances) from a fixed input instance. In fact, they can also be viewed as transforming a probabilistic input database to a probabilistic output database. Such transformations between probability spaces—assuming they satisfy certain technical conditions—are known as *stochastic kernels*.

Our semantics essentially extends the semantics of Bárány et al. [3] for discrete distributions to continuous ones, which requires a more principled approach rooted in tools from measure theory. The basic idea of our semantics is the same as in [3]: we associate an *existential* Datalog program with a every GDatalog program, define a probability distribution on the paths of a chase tree for this program, and then derive a distribution on database instances from this distribution on the chase tree. To define a distribution on the chase tree, we show that each chase step is a stochastic kernel between probabilistic databases. This allows viewing the chase tree as a Markov process. The probability distribution we associate with the chase tree is the so-called push-forward measure of the Markov process. A technical difficulty we have to deal with is that only finite paths in a chase tree actually correspond to database instances, because instances are always required to be finite.

Our main technical result is a proof that the semantics is independent of the choice of the chase tree and that it is equivalent to the semantics obtained from parallel execution of all applicable rules at any step of the execution of a Datalog program.

We slightly modify the semantics of Bárány et al. [3] even for discrete distributions. The reason is that we want to avoid certain peculiarities of the original semantics, illustrated in the example below. We note, however, that apart from this, our change in the semantics is not central for the mathematical developments of this paper.

Example 1.1. Consider the GDatalog programs $\mathcal{G}_0$ and $\mathcal{G}_\varepsilon$:

$$\boxed{\begin{aligned}\mathcal{G}_0\colon\ & R(\mathsf{Flip}\langle 1/2\rangle) \leftarrow \top \\ & R(\mathsf{Flip}\langle 1/2\rangle) \leftarrow \top\end{aligned}} \qquad \boxed{\begin{aligned}\mathcal{G}_\varepsilon\colon\ & R(\mathsf{Flip}\langle 1/2\rangle) \leftarrow \top \\ & R(\mathsf{Flip}\langle 1/2+\varepsilon\rangle) \leftarrow \top\end{aligned}}$$

For $p \in [0,1]$, the rule $R(\mathsf{Flip}(p))$ generates the fact $R(1)$ with probability p and the fact $R(0)$ with probability $1-p$.

Under the semantics of [3], the program $\mathcal{G}_0$ generates with probability $1/2$ the instance $\{R(1)\}$ and with probability $1/2$ the instance $\{R(0)\}$. For $0 < \varepsilon \leq 1/2$, the program $\mathcal{G}_\varepsilon$ generates with probability $1/4 + \varepsilon + \varepsilon^2$ the instance $\{R(1)\}$, with probability $1/4 - \varepsilon + \varepsilon^2$ the instance $\{R(0)\}$, and with probability $1/2 - 2\varepsilon^2$ the instance $\{R(1), R(0)\}$. Thus, contrary to the intuition, the outcome of $\mathcal{G}_\varepsilon$ *does not converge* to that of $\mathcal{G}_0$ as $\varepsilon \to 0$.

Under our semantics, the outcome of $\mathcal{G}_\varepsilon$ remains as described above, whereas the outcome of $\mathcal{G}_0$ becomes $\{R(1)\}$ or $\{R(0)\}$, each with probability $1/4$, and $\{R(1), R(0)\}$ with probability $1/2$.

As another example, consider the following program $\mathcal{G}_0'$:

$$\boxed{\begin{aligned}\mathcal{G}_0'\colon\ & R(\mathsf{Flip}\langle 1/2\rangle) \leftarrow \top \\ & R(\mathsf{Flip}'\langle 1/2\rangle) \leftarrow \top\end{aligned}}$$

Here Flip' is the same Bernoulli distribution as Flip, but with a different name. Then, under the semantics of [3], the outcome of $\mathcal{G}_0'$ is $\{R(1)\}$ or $\{R(0)\}$, each with probability $1/4$, and $\{R(1), R(0)\}$ with probability $1/2$. Note that this differs from the outcome of the, intuitively equivalent, program $\mathcal{G}_0$. Under our semantics, the outcomes of $\mathcal{G}_0$ and $\mathcal{G}_0'$ are the same.

We believe that our semantics is simpler and arguably more intuitive. On the downside, in our semantics, programs are no longer invariant under first-order equivalence when random terms are just interpreted as terms using function symbols. This difference in the semantics is inessential to the goals of this paper however, because we can rewrite programs in a way such that our semantics simulates the semantics of [3] and vice versa.

Paper Outline. This paper is organized as follows. After concluding the introduction with a short survey of related work, we present the central mathematical definitions and background results from measure theory and probabilistic databases in Section 2. In Section 3, we introduce the syntax of GDatalog programs together with the backbone of its semantics, that is, the translation into an existential Datalog program. In Section 4 we present our version of a probabilistic chase, generalizing the ideas of [3]. We show that this notion defines a Markov process over database instances. Section 5 is devoted to establishing similar results when a parallel chase procedure is used (which is novel over [3]). In Section 6, we discuss various properties of the semantics. First, we show that no matter which kind of chase procedure is used, the probabilistic database that is described by its semantics turns out to be the same. We argue that our semantics can simulate the original semantics of [3] and look the termination behavior of GDatalog programs. We conclude the work and indicate topics for future research in Section 7.

Related Work. In both the fields of probabilistic programming as well as probabilistic databases, there is a variety of models and systems that allow to specify continuous probability distributions. We will mention some of them and indicate how they compare to the scope of this paper. Note that the original work on Probabilistic Programming Datalog [3] contains a broad discussion of related concepts.

Several "pure" programming languages support continuous distributions, for instance Church [17], Anglican [39], and Figaro [33]. Languages that are conceptually closer to Probabilistic Programming Datalog are those with direct ties to *statistical relational artificial intelligence (StarAI)* [10]. A prominent example of such a language is ProbLog [9], a probabilistic variant of Prolog. In [23], Problog has been extended by continuous distributions. Formalisms such as Markov Logic Networks (MLNs) [34] (which have also been equipped with continuous semantics [36]) have closer ties to so-called *probabilistic graphical models* [27]. Probabilistic Soft Logic (PSL) uses weighted Horn clauses as a "frontend" for graphical models [26]. Similarly, as Markov Logic Networks have Markov Networks as their backbone, the language BLOG [31] builds upon

Bayesian networks. Recently, its continuous semantics have received a thorough measure-theoretic treatment [42].

While all languages and formalisms mentioned above share individual features with Probabilistic Programming Datalog, conjoining Datalog with classical probabilistic programming was novel to [3]. Earlier proposed probabilistic versions of Datalog could, for example specify a prior on the data [14] or let rules fire probabilistically [11]. The probabilistic Datalog version JudgeD [41] supports the introduction of dependencies among rules and facts using annotations with logical formulae. Another probabilistic Datalog language that introduces randomness by "event annotations" was introduced in [19] where the language is used for specifying *ontologies*.

Finally, we mention MCDB [24] and its successor SimSQL [6]. Here, users are able to specify probabilistic models in the shape of random database instances. In particular, SimSQL can define Markov processes over database instances.

2 PRELIMINARIES

2.1 Foundations from Measure Theory

We briefly recall measure theoretical notions needed for our development. For more details, we refer to Kallenberg [25] and Srivastava [37]. Appendix B contains some well-known key results that we do not state here for space reasons.

2.1.1 Measure Spaces. A family $\mathfrak{X}$ of subsets of a set $\mathbf{X}$ is called a *σ-algebra* on $\mathbf{X}$ if it contains $\mathbf{X}$ and is closed under relative complementation w. r. t. $\mathbf{X}$ and under countable unions. A pair $(\mathbf{X}, \mathfrak{X})$ with $\mathfrak{X}$ being a σ-algebra on $\mathbf{X}$ is called a *measurable space*. The elements of $\mathfrak{X}$ are called *measurable sets* or *events*.

Let $\mathfrak{G}$ be a family of subsets of $\mathbf{X}$. Then $\sigma(\mathfrak{G})$ denotes the coarsest σ-algebra on $\mathbf{X}$ containing $\mathfrak{G}$, i. e. the intersections of all σ-algebras on $\mathbf{X}$ containing $\mathfrak{G}$. We say $\sigma(\mathfrak{G})$ is *generated* by $\mathfrak{G}$.

If $(\mathbf{X}, \mathfrak{X})$ is a measurable space and $\mathcal{X} \subseteq \mathbf{X}$, then $\mathfrak{X} \restriction \mathcal{X} := \{\mathcal{X}' \cap \mathcal{X} \colon \mathcal{X}' \in \mathfrak{X}\}$ is a σ-algebra on $\mathbf{X} \cap \mathcal{X}$, called the *trace σ-algebra* of $\mathcal{X}$.

Another standard construction is the *disjoint union σ-algebra*. Let $(\mathbf{X}, \mathfrak{X})$ and $(\mathbf{Y}, \mathfrak{Y})$ be measurable spaces with $\mathbf{X} \cap \mathbf{Y} = \emptyset$. Then the family $\mathfrak{X} \oplus \mathfrak{Y}$, defined by

$$\mathcal{Z} \in \mathfrak{X} \oplus \mathfrak{Y} :\Leftrightarrow \mathcal{Z} \cap \mathbf{X} \in \mathfrak{X} \text{ and } \mathcal{Z} \cap \mathbf{Y} \in \mathfrak{Y}$$

is a σ-algebra on $\mathbf{X} \uplus \mathbf{Y}$. This generalizes to more than two "summands" in a straight-forward manner [12, § 214L]. For I being some finite set of indices, we denote the disjoint union σ-algebra of (suitable) σ-algebras $\mathfrak{X}_i$, $i \in I$ by $\bigoplus_{i \in I} \mathfrak{X}_i$.

A function $\mu \colon \mathfrak{X} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a *measure* on a measurable space $(\mathbf{X}, \mathfrak{X})$ if $\mu(\bigcup_{i \in \mathbb{N}} \mathcal{X}_i) = \sum_{i \in \mathbb{N}} \mu(\mathcal{X}_i)$ for any sequence of pairwise disjoint events $\mathcal{X}_i \in \mathfrak{X}$ ($i \in \mathbb{N}$). An event $\mathcal{X} \in \mathfrak{X}$ is said to be *of (μ-)measure* $\mu(\mathcal{X})$. The value $\mu(\mathbf{X})$ is called the *mass* of μ. Measures of total mass $\mu(\mathbf{X}) = 1$ are called *probability measures*, measures of mass $\mu(\mathbf{X}) \leq 1$ are called *sub-probability measures*.

A triple $(\mathbf{X}, \mathfrak{X}, \mu)$ is called a *measure space* if $(\mathbf{X}, \mathfrak{X})$ is a measurable space and μ is a measure on $(\mathbf{X}, \mathfrak{X})$. The measure space $(\mathbf{X}, \mathfrak{X}, \mu)$ (or simply μ) is called σ-finite, if there exists a partition of $\mathbf{X}$ into countably many measurable sets of finite measure. All kinds of measures that appear in this paper are σ-finite.

A *topological space* is a pair $(\mathbf{X}, \mathfrak{T})$ where $\mathbf{X}$ is a set and $\mathfrak{T}$ is a family of subsets of $\mathbf{X}$, called the *open sets*, such that $\mathfrak{T}$ contains both $\mathbf{X}$ and $\emptyset$ and is closed under *finite* intersections and *arbitrary* unions. The σ-algebra on a topological space $(\mathbf{X}, \mathfrak{T})$ that is generated by the open sets is called the *Borel σ-algebra* of $(\mathbf{X}, \mathfrak{T})$ (resp. on $\mathbf{X}$ if $\mathfrak{T}$ is understood from context). We denote the Borel σ-algebra on $\mathbf{X}$ by $\mathcal{B}or(\mathbf{X})$. Typical examples are $\mathcal{B}or(\mathbb{R})$ and $\mathcal{B}or[0,1] := \mathcal{B}or([0,1])$.

A central object for modern probability theory are the Borel σ-algebras generated from *Polish* topological spaces, i.e. from completely metrizable spaces containing a countable dense set. The resulting measurable spaces are called *standard Borel spaces*. We do not delve into the details here, as all measurable spaces appearing in this paper are standard Borel. For further information, especially in the context of probabilistic databases, see [22].

2.1.2 Measurable Functions and Kernels. Let $(\mathbf{X}, \mathfrak{X})$ and $(\mathbf{Y}, \mathfrak{Y})$ be measurable spaces. A function $f \colon \mathbf{X} \to \mathbf{Y}$ is called $(\mathfrak{X}, \mathfrak{Y})$-*measurable* (or simply *measurable*, if clear from context) if the preimage of every measurable set in $\mathbf{Y}$ is measurable in $\mathbf{X}$; that is, if $f^{-1}(\mathcal{Y}) := \{X \in \mathbf{X} \colon f(X) \in \mathcal{Y}\} \in \mathfrak{X}$ whenever $\mathcal{Y} \in \mathfrak{Y}$.

If f, as above, is $(\mathfrak{X}, \mathfrak{Y})$-measurable and μ is a measure on $(\mathbf{X}, \mathfrak{X})$, then $\mu \circ f^{-1}$ is the so-called *push-forward measure* of μ along f on $(\mathbf{Y}, \mathfrak{Y})$. If μ is a (sub-)probability measure, so is $\mu \circ f^{-1}$.

A function $\kappa \colon \mathbf{X} \times \mathfrak{Y} \to [0,1]$ is called a *(sub-)stochastic kernel* from $(\mathbf{X}, \mathfrak{X})$ to $(\mathbf{Y}, \mathfrak{Y})$ if

- f. a. $X \in \mathbf{X}$, $\kappa(X, \cdot) \colon \mathfrak{Y} \to [0,1]$ is a (sub-)probability measure on $(\mathbf{Y}, \mathfrak{Y})$,
- f. a $\mathcal{Y} \in \mathfrak{Y}$, $\kappa(\cdot, \mathcal{Y}) \colon \mathbf{X} \to [0,1]$ is $(\mathfrak{X}, \mathcal{B}or[0,1])$-measurable.

For every measurable space $(\mathbf{X}, \mathfrak{X})$, the function $\iota \colon \mathbf{X} \times \mathfrak{X} \to [0,1]$ with $\iota(X, \mathcal{X}) = 1$ if $X \in \mathcal{X}$ and $\iota(X, \mathcal{X}) = 0$ if $X \notin \mathcal{X}$ is a stochastic kernel from $(\mathbf{X}, \mathfrak{X})$ to itself, called the *identity kernel* on $(\mathbf{X}, \mathfrak{X})$.

2.1.3 Product Measures. Let $(\mathbf{X}_i, \mathfrak{X}_i)$, with $i \in I$ for some index set I, be a collection of measurable spaces and let $\mathbf{X} := \prod_{i \in I} \mathbf{X}_i$. The *product σ-algebra* $\bigotimes_{i \in I} \mathfrak{X}_i$ is the coarsest σ-algebra on $\mathbf{X}$ that makes all canonical projections $\pi_i \colon \mathbf{X} \to \mathbf{X}_i \colon (x_i)_{i \in I} \mapsto x_i$ measurable. If I is countable, then $\bigotimes_{i \in I} \mathfrak{X}_i$ is generated by the family of *measurable rectangles* $\prod_{i \in I} \mathcal{X}_i$ with $\mathcal{X}_i \in \mathfrak{X}_i$. If $I = \{1, \ldots, n\}$ we write $\bigotimes_{i=1}^{n} \mathfrak{X}_i$ or $\mathfrak{X}_1 \otimes \cdots \otimes \mathfrak{X}_n$ for the product σ-algebra. If all $(\mathbf{X}_i, \mathfrak{X}_i)$ are equal, we write $\mathfrak{X}^{\otimes n}$. For I countable, we write $\mathfrak{X}^{\otimes \omega}$.

If $f \colon \mathbf{X} \to \mathbf{Y}$ is measurable with $(\mathbf{X}, \mathfrak{X})$ and $(\mathbf{Y}, \mathfrak{Y})$ standard Borel, then the *graph of f*, defined by

$$\mathsf{graph}(f) := \{(x, f(x)) \colon x \in \mathbf{X}\} \in \mathbf{X} \times \mathbf{Y},$$

is measurable in $\mathfrak{X} \otimes \mathfrak{Y}$ [37, Proposition 3.1.21 and 2.1.9].

If $\mathcal{Z} \subseteq \mathbf{X} \times \mathbf{Y}$ and $x \in \mathbf{X}$, then $\mathcal{Z}_x := \{y \in \mathbf{Y} \colon (x, y) \in \mathcal{Z}\} \in \mathbf{Y}$ is called the *x-section* of $\mathcal{Z}$. If $\mathcal{Z} \in \mathfrak{X} \otimes \mathfrak{Y}$, then $\mathcal{Z}_x \in \mathfrak{Y}$. The same applies symmetrically for the y-section $\mathcal{Z}_y$ with $y \in \mathbf{Y}$, i. e. $\mathcal{Z}_y \in \mathfrak{X}$.

If $(\mathbf{X}, \mathfrak{X}, \mu)$ and $(\mathbf{Y}, \mathfrak{Y}, \nu)$ are measure spaces with μ and ν σ-finite, then there exists a unique *product measure* $\mu \otimes \nu$ of μ and ν on $(\mathbf{X} \times \mathbf{Y}, \mathfrak{X} \otimes \mathfrak{Y})$ with the property that $(\mu \otimes \nu)(\mathcal{X} \times \mathcal{Y}) = \mu(\mathcal{X}) \cdot \nu(\mathcal{Y})$ for all $\mathcal{X} \in \mathfrak{X}$ and $\mathcal{Y} \in \mathfrak{Y}$. This can be extended to any finite (nonempty) product of measures [25, cf. Theorem 1.27 and p. 15]. We use the notation $\bigotimes_{i=1}^{n} \mu_i$ and $\mu^{\otimes n}$ analogous to the one for product σ-algebras.

By Fubini's Theorem (Fact B.8, see [25, Theorem 1.27]), for every measurable $f \colon \mathbf{X} \times \mathbf{Y} \to \mathbb{R}_{\geq 0}$ it holds that

$$\int f \; d(\mu \otimes \nu) = \int \int f \; d\mu \; d\nu = \int \int f \; d\nu \; d\mu$$

whenever μ and ν are σ-finite.

2.1.4 Multifunctions and Selections. Let $(\mathbf{X}, \mathfrak{X})$ and $(\mathbf{Y}, \mathfrak{Y})$ be measurable spaces where $(\mathbf{Y}, \mathfrak{Y})$ is standard Borel (with fixed Polish topology $\mathfrak{T}_\mathbf{Y}$), and let $2^\mathbf{Y}$ denote the power set of $\mathbf{Y}$. A function $M\colon \mathbf{X} \to 2^\mathbf{Y} \setminus \emptyset$ is called a *multifunction*. We write $M\colon \mathbf{X} \rightrightarrows \mathbf{Y}$ if M is a multifunction from $\mathbf{X}$ to $\mathbf{Y}$. A multifunction $M\colon \mathbf{X} \rightrightarrows \mathbf{Y}$ is called

- *closed-valued*, if for every $x \in \mathbf{X}$, $M(x) \subseteq \mathbf{Y}$ is closed w. r. t. $\mathfrak{T}_\mathbf{Y}$, and
- $\mathfrak{X}$-*measurable*, if $M^{-1}(\mathcal{Y}) \coloneqq \{x \in \mathbf{X}\colon M(x) \cap \mathcal{Y} \neq \emptyset\} \in \mathfrak{X}$ for every open set $\mathcal{Y} \in \mathfrak{T}_\mathbf{Y}$.

Similarly to the corresponding statement for measurable functions, if $M\colon \mathbf{X} \rightrightarrows \mathbf{Y}$ is a closed-valued measurable multifunction, then

$$\mathsf{graph}(M) \coloneqq \{(x, y)\colon y \in M(x)\} \in \mathbf{X} \times \mathbf{Y}$$

is a measurable set in $\mathfrak{X} \otimes \mathfrak{Y}$.

A *selection* of a multifunction M is a function $s\colon \mathbf{X} \to \mathbf{Y}$ with $s(x) \in M(x)$ for all $x \in \mathbf{X}$. A well-known result from Kuratowski and Ryll-Nardzewski (Fact B.5, see [29] and [37, Theorem 5.2.1]) states that for $(\mathbf{Y}, \mathfrak{Y})$ standard Borel, every measurable, closed-valued multifunction $M\colon \mathbf{X} \rightrightarrows \mathbf{Y}$ has a $(\mathfrak{X}, \mathfrak{Y})$-measurable selection.

2.1.5 (Discrete-Time) Stochastic Processes. A *stochastic process in discrete time* is basically a sequence of random elements over some *state space* $(\mathbf{X}, \mathfrak{X})$. Intuitively, a (discrete-time) *Markov process* is a stochastic process where the distribution in the ith step only depends on the distribution of the previous step $i-1$. By a theorem of Kolmogorov (Fact B.9), Markov processes in discrete time are guaranteed to exist for any initial distribution and any sequence of stochastic kernels κ_i where κ_i describes the probabilistic transition of the ith step of the process. If $(\mathbf{X}, \mathfrak{X})$ is the state space of the process, then $(\mathbf{X}^\omega, \mathfrak{X}^{\otimes\omega})$ is its *path space*.

2.2 Parameterized Distributions

In the extension of conventional Datalog that we consider in this paper, so-called *parameterized distributions* occur.

Definition 2.1. A *parameterized distribution* consists

(i) of a measure space $(\mathbf{X}, \mathfrak{X}, \mu)$ such that either
- $(\mathbf{X}, \mathfrak{X}, \mu)$ is the Euclidean space $\mathbb{R}^d$ with its Lebesgue-measurable sets and μ is the (d-dimensional) Lebesgue measure; or
- $\mathbf{X}$ is discrete, $\mathfrak{X} = 2^\mathbf{X}$ and μ is the counting measure on $(\mathbf{X}, \mathfrak{X})$ that maps every subset of $\mathbf{X}$ to its cardinality;

(ii) and a function $\psi\colon \Theta \times \mathbf{X} \to \mathbb{R}_{\geq 0}$ such that
- $\psi(\theta, \cdot)$ is a measurable function for all $\theta \in \Theta$, and
- the property that $\int_\mathbf{X} \psi(\theta, \cdot)\ d\mu = 1$ for all $\theta \in \Theta$.

We usually identify a parameterized distribution via ψ with $(\mathbf{X}, \mathfrak{X}, \mu)$ and Θ left implicit if not specified otherwise. In this case, we usually refer to $\mathbf{X}$ as $\mathsf{dom}\,\psi$, to μ as μ_ψ and to Θ as Θ_ψ. Moreover, we usually make it explicit in the notation which part of the expression $\psi(\theta, x)$ forms the parameter by writing $\psi\langle\theta\rangle(x)$ instead.

Note that for a parameterized distribution ψ and every fixed $\theta \in \Theta_\psi$ the function

$$P_{\psi\langle\theta\rangle}\colon \mathfrak{X}_\psi \to [0,1]\colon \mathcal{X} \mapsto \int_\mathcal{X} \psi\langle\theta\rangle\ d\mu_\psi \tag{2.A}$$

is a probability measure on $\mathbf{X}_\psi$. In particular, if $\mathbf{X}_\psi$ is the real line, $\psi\langle\theta\rangle$ is a *probability density function*. If $\mathbf{X}_\psi$ is discrete, then $\psi\langle\theta\rangle$ is a *probability mass function* and $\psi\langle\theta\rangle = P_{\psi\langle\theta\rangle}$ with our definition from above. In any case, if $P_{\psi\langle\theta\rangle}$ is among the "typical" probability distributions, we will refer to the parameterized distribution by a symbolic name such as Binomial, Poisson or Normal.

Example 2.2. Consider a set $\Psi = \{\mathsf{Flip}, \mathsf{Binomial}, \mathsf{Poisson}, \mathsf{Normal}\}$ of parameterized distributions.

- The Flip distribution models a coin flip that can be parameterized with the coin's bias. Thus, $\Theta_\mathsf{Flip} = [0,1]$ and $\mathsf{Flip}\langle\theta\rangle(1) = \theta$ and $\mathsf{Flip}\langle\theta\rangle(0) = 1-\theta$.
- The Binomial distribution is a discrete distribution of finite support for all individual parameters. Its parameters are $\Theta_\mathsf{Binomial} = \{(n,k) \in \mathbb{N}^2\colon k \leq n\}$. Note however, that the union of all supports over all possible parameters is infinite.
- The Poisson distribution is discrete, although here, the support is infinite for any fixed parameter θ. We have $\Theta_\mathsf{Poisson} = \mathbb{R}_{>0}$ and $\mathsf{Poisson}\langle\lambda\rangle(k) = \lambda^k e^{-\lambda}/k!$.
- The normal distribution Normal is an "absolutely continuous" (see [25, p. 29]) distribution with $\Theta_\mathsf{Normal} = \mathbb{R} \times \mathbb{R}_{>0}$ and

$$\mathsf{Normal}\langle\mu, \sigma^2\rangle(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{\sigma^2}}.$$

In our application we need to be able to handle probability distributions over parametrizations for a parameterized distribution. Thus, the following result on measurability with respect to parametrizations is central for our work. It is a special case of [15, Theorem 3.2], tailored to our definition of parameterized distributions. It states that, under suitable technical conditions, the probability of a fixed event under a parameterized distribution is a measurable function of the parameters.

Fact 2.3 (Gaudard & Hadwin [15, Thm. 3.2])**.** *Let ψ be a parameterized distribution on $(\mathbf{X}, \mathfrak{X}, \mu)$ as in Definition 2.1 with $\Theta \coloneqq \Theta_\psi$ being a Borel subset of a Polish space, such that:*

- *for every $x \in \mathbf{X}$, the function $\Theta \to [0,1]\colon \theta \mapsto \psi\langle\theta\rangle(x)$ is continuous; and*
- *every $\theta_0 \in \Theta$ has a neighborhood $N(\theta_0)$ with*

$$\int_\mathbf{X} \left(\sup_{\theta \in N(\theta_0)} \psi\langle\theta\rangle \right) d\mu < \infty; \quad \textit{and}$$

- *if $\theta, \theta' \in \Theta_\psi$ with $\theta \neq \theta'$, then $P_{\psi\langle\theta\rangle}$ and $P_{\psi\langle\theta'\rangle}$ are different probability measures.*

Then for every $\mathcal{X} \in \mathfrak{X}$ and $\mathcal{Y} \in \mathcal{B}or[0,1]$, it holds that

$$\left\{\theta \in \Theta\colon P_{\psi\langle\theta\rangle}(\mathcal{X}) \in \mathcal{Y}\right\} \in \mathcal{B}or(\Theta_\psi)$$

with $P_\psi\langle\theta\rangle$ as in Eq. (2.A).

As stated in [15], the previous fact applies to "most common parametric families" [15, p. 173], including, for example, all the distributions of Example 2.2.

Remark 2.4. For our technical developments in the main part of the paper, we might also consider using distributions that are mixtures of discrete and continuous measures. The corresponding proofs might then be carried out by considering these parts separately.

2.3 Probabilistic Databases

In a nutshell, a probabilistic database (PDB) is a collection of traditional database instances that is equipped with a probability measure. Throughout this paper, we use the framework of *standard probabilistic databases* as developed in [22]. We will only briefly introduce the important notions here and refer the reader to [22] for details of the construction. For a database schema $\mathcal{S}$, we let $\mathbb{F}_{\mathcal{S}}$ denote the set of facts that can be built from $\mathcal{S}$. A basic assumption for standard PDBs is that all attribute domains are standard Borel. Then $\mathbb{F}_{\mathcal{S}}$ is standard Borel as well and we denote its (Borel) σ-algebra by $\mathfrak{F}_{\mathcal{S}}$. The sample space $\mathbb{D}$ of a standard PDB is then the set of all finite bags of facts. By a generic construction, $\mathbb{D}$ is equipped with a σ-algebra $\mathfrak{D}$, turning it into a measurable space. The σ-algebra $\mathfrak{D}$ is generated by the family of *counting events* $C(\mathcal{F}, n)$ consisting of those instances that contain exactly n facts from $\mathcal{F}$, where $\mathcal{F}$ is a measurable set of facts. Any probability measure P on $(\mathbb{D}, \mathfrak{D})$ then yields a *(standard) probabilistic database* $\Delta = (\mathbb{D}, \mathfrak{D}, P)$. As we only work with standard PDBs, we omit the term "standard" henceforth.

Fact 2.5 (see Theorem A.1)**.** *The measurable space of standard PDBs is a standard Borel space.*

Fact 2.6 (Measurability of Queries[22])**.** *Relational algebra and aggregate queries are measurable functions on PDBs.*

The construction of PDBs sketched before inherently uses bag semantics stemming from its mathematical backbone of *(finite) point processes*. For the purpose of this paper, we only want to consider set semantics though. This can either be achieved on the side of measures, i. e. PDBs with almost surely set-valued instances; or by restricting the sample space to the set $\mathbb{D}_*$ of duplicate-free instances from $\mathbb{D}$. Note that $\mathbb{D}_*$ is a measurable subset of $\mathbb{D}$ and, consequentially, $\mathfrak{D}_* := \mathfrak{D} \restriction \mathbb{D}_*$ a sub-σ-algebra of $\mathfrak{D}$. Moreover, $\mathfrak{D}_*$ is generated by the family of all set-valued counting events $C_*(\mathcal{F}, n) := C(\mathcal{F}, n) \cap \mathbb{D}_*$ (cf. [37, p. 83]).

Throughout this paper we will exclusively use set instances and set semantics. To simplify notation, we write $(\mathbb{D}, \mathfrak{D})$ *instead of* $(\mathbb{D}_*, \mathfrak{D}_*)$ *for the measurable space of set instances.*

Definition 2.7 (Subprobabilistic Databases)**.** Let $\Delta = (\mathbb{D}, \mathfrak{D}, P)$ be a PDB and let $\alpha \in [0, 1]$. Then the measure space $\Delta_\alpha := (\mathbb{D}, \mathfrak{D}, p)$ with the sub-probability measure $p = \alpha P$ is called *subprobabilistic database (SubPDB)*. The value α is called the *mass* of the SubPDB Δ_α.

Let $\mathbb{D}_{\mathsf{err}} := \mathbb{D} \uplus \{\mathsf{err}\}$ and equip this space with the σ-algebra $\mathfrak{D}_{\mathsf{err}} := \mathfrak{D} \cup \{\mathcal{D} \cup \{\mathsf{err}\} \colon \mathcal{D} \in \mathfrak{D}\}$. Every probability measure P on $(\mathbb{D}_{\mathsf{err}}, \mathfrak{D}_{\mathsf{err}})$ yields a sub-probability measure p on $(\mathbb{D}, \mathfrak{D})$ and vice versa such that $P(\mathcal{D}) = p(\mathcal{D})$ for all $\mathcal{D} \in \mathfrak{D}$.

A natural interpretation of the "missing" probability mass $1 - \alpha$ of an SubPDB is that it describes the probability of an error event (or the outcome of a draw from the PDB to be undefined). The space $\mathbb{D}_{\mathsf{err}}$ makes this error event ("err") explicit.

Note that the results of [22] concerning query measurability directly also apply to subprobabilistic databases.

3 GENERAL GENERATIVE DATALOG

We assume the reader is familiar with the usual syntax and semantics of conventional Datalog programs. The syntax of *Generative Datalog* consists of components intended to carry both traditional logical and stochastic semantics.

3.1 Logical Setup of Generative Datalog Rules

Let $\mathcal{I}$ and $\mathcal{E}$ be two disjoint relational schemas, called the *intensional* and the *extensional* schemas, respectively. Let $\mathbb{V}$ be a countably infinite set of variables. We fix a family Ψ of parameterized distributions such that Θ_ψ is a standard Borel space for all $\psi \in \Psi$.

Definition 3.1 (Terms)**.** All variables $x \in \mathbb{V}$ and all constants c from $\mathcal{I} \cup \mathcal{E}$ are *deterministic terms*.

Let $\psi \in \Psi$ and let θ be a tuple of constants and variables such that there exists a valuation of the variables that maps θ into Θ_ψ. Then $\psi\langle\theta\rangle$ is a *random term*.

Definition 3.2 (Atoms)**.** An *$\mathcal{I}$-atom* (resp. *$\mathcal{E}$-atom*) is an expression of the form $R(t_1, \ldots, t_n)$ where n is the arity of $R \in \mathcal{I}$ (resp. $R \in \mathcal{E}$) and $t_1, \ldots, t_n$ are terms. We require that

- if $t_i = c$ is a constant, then it is a constant in the attribute domain of the respective attribute in R; and
- if $t_i = \psi\langle\theta\rangle$ is a random term, then $R \in \mathcal{I}$ and $\mathbf{X}_\psi$ is contained in the attribute domain of the respective attribute in R.

If at least one of the terms t_i is random, $R(t_1, \ldots, t_n)$ is called a *random atom*; otherwise, it is called a *deterministic atom*.

Definition 3.3 (GDatalog Rules, cf. [3, Definition 3.1])**.** A *GDatalog rule* φ is an expression of the shape $\varphi = \varphi_h(\bar{x}) \leftarrow \varphi_b(\bar{x})$ where

- $\varphi_h(\bar{x})$ is an $\mathcal{I}$-atom (called the *head* of the rule) where the free variables of φ_h are a subset of the variables appearing in $\bar{x}$; and
- $\varphi_b(\bar{x})$ is a conjunction of deterministic $(\mathcal{I} \cup \mathcal{E})$-atoms having exactly $\bar{x}$ as its free variables (called the *body* of the rule).

A rule φ is called *random*, if it contains a random atom, and *deterministic* otherwise. A *GDatalog program* $\mathcal{G}$ is a finite collection of GDatalog rules. As commonly done, we denote the conjunctions within the rule body by commas.

Example 3.4. Below, we express the example from [3, Figure 3, p. 22:8] in our syntax. (Its intended meaning is ibidem described in detail). Here, $\Psi = \{\mathsf{Flip}\}$ as described in Example 2.2.

$$
\begin{aligned}
\mathcal{G}\colon\ & \mathsf{Earthquake}(c, \mathsf{Flip}\langle 0.1\rangle) \leftarrow \mathsf{City}(c, r)\\
& \mathsf{Unit}(h, c) \leftarrow \mathsf{House}(h, c)\\
& \mathsf{Unit}(b, c) \leftarrow \mathsf{Business}(b, c)\\
& \mathsf{Burglary}(x, c, \mathsf{Flip}\langle r\rangle) \leftarrow \mathsf{Unit}(x, c), \mathsf{City}(c, r)\\
& \mathsf{Trig}(x, \mathsf{Flip}\langle 0.6\rangle) \leftarrow \mathsf{Unit}(x, c), \mathsf{Earthquake}(c, 1)\\
& \mathsf{Trig}(x, \mathsf{Flip}\langle 0.9\rangle) \leftarrow \mathsf{Burglary}(x, c, 1)\\
& \mathsf{Alarm}(x) \leftarrow \mathsf{Trig}(x, 1)
\end{aligned}
$$

Example 3.5. The following is a simple example program using the (continuous) normal distribution. PCountry is a list of persons together with their home country and CMoments contains the expectation and the variance of people's heights within the country c. The program shall construct a list of persons with their height

from this data by random sampling.

$$\mathcal{G}\colon\ \mathrm{PHeight}(p, \mathrm{Normal}\langle \mu, \sigma^2\rangle) \ \leftarrow\ \mathrm{PCountry}(p, c),\ \mathrm{CMoments}(c, \mu, \sigma^2)$$

3.2 Associating Existential Datalog Programs

Having introduced the syntax of GDatalog programs, we now proceed with the mathematical description of their semantics. For that, we adopt the idea of [3] to associate an existential Datalog program to every GDatalog program. An *existential Datalog program* (Datalog$^{\exists}$ program) is a GDatalog program without random atoms that additionally allows rules of the shape $\exists y\colon \varphi_h(\bar{x}, y) \leftarrow \varphi_b(\bar{x})$. The semantics of GDatalog is then given in terms of its associated Datalog$^{\exists}$ program. *To simplify the proofs, we will assume that every probabilistic rule of a GDatalog program contains* exactly one *parameterized distribution. Our proofs can be generalized to multiple parameterized distributions though, using their product densities.*

Let $\mathcal{G} = \{\varphi_1, \dots, \varphi_k\}$ be a GDatalog program and let $\varphi_i(\bar{x}) = \varphi_{i,h}(\bar{x}) \leftarrow \varphi_{i,b}(\bar{x})$ be a rule in $\mathcal{G}$. (Recall that $\varphi_{i,h}(\bar{x})$ means that $\varphi_{i,h}$ has free variables *among* $\bar{x}$, not necessarily exactly $\bar{x}$.) If φ_i is deterministic, we leave it unchanged. So suppose φ_i is a random rule, say, with head $\varphi_{i,h}(\bar{x}) = R(x_1, \dots, x_n, \psi\langle p_1, \dots, p_m\rangle)$ where every x_ℓ, $1 \le \ell \le n$ is either a variable from $\bar{x}$ or a constant and likewise for p_ℓ, $1 \le \ell \le m$. In this case, we replace φ_i with the following two rules:

$$\exists y\colon R_i(x_1, \dots, x_n, p_1, \dots, p_m, y) \ \leftarrow\ \varphi_{i,b}(\bar{x}) \tag{3.A}$$

$$R(x_1, \dots, x_n, y) \ \leftarrow\ \varphi_{i,b}(\bar{x}), R_i(x_1, \dots, x_n, p_1, \dots, p_m, y) \tag{3.B}$$

Where R_i is a new, distinguished relation symbol. This procedure associated a Datalog$^{\exists}$ program $\hat{\mathcal{G}}$ to the GDatalog program $\mathcal{G}$. We call the rules of the shape of Eq. (3.A) *existential*. All other rules of $\hat{\mathcal{G}}$ are called *deterministic*.

3.3 Rule Applicability

In this subsection, we formalize the notion of rules being applicable for valuations of their free variables in a measure-theoretic way. A rule being applicable for a valuation in an instance intuitively means that it is allowed to fire for that instance (and the valuation) in the execution of the program.

Let $\hat{\varphi}$ be a rule of the Datalog$^{\exists}$ program $\hat{\mathcal{G}}$ associated to our GDatalog program $\mathcal{G}$. For the tuple $\bar{x}$ of free variables of $\hat{\varphi}$, we let $\mathbb{V}_{\hat{\varphi}}$ be their associated joint domain, i. e. the Cartesian product of the attribute domains for the positions in the relation symbols where the free variables occur.[1] The space $\mathbb{V}_{\hat{\varphi}}$ is equipped with its product σ-algebra $\mathfrak{V}_{\hat{\varphi}}$.

A pair $(\hat{\varphi}, \bar{a})$ with $\hat{\varphi} \in \hat{\mathcal{G}}$ and $\bar{a} \in \mathbb{V}_{\hat{\varphi}}$ is called *applicable* in D if $D \not\models \hat{\varphi}_h(\bar{a})$ and $D \models \hat{\varphi}_b(\bar{a})$. (That is, D satisfies the rule body, but not the head.) The reason to explicitly require that the head is not fulfilled comes from the semantics of existential Datalog programs. We let $\mathsf{App}(D)$ denote the set of applicable pairs in D. If there is no pair applicable in D, we set $\mathsf{App}(D) = \{(\square, \square)\}$. Formally, this defines a multifunction $\mathsf{App}\colon \mathbb{D} \rightrightarrows \mathbb{A}$ where

$$\mathbb{A} \coloneqq \left\{(\hat{\varphi}, \bar{a})\colon \hat{\varphi} \in \hat{\mathcal{G}} \text{ and } \bar{a} \in \mathbb{V}_{\hat{\varphi}}\right\} \cup \{(\square, \square)\}.$$

Note that for all $D \in \mathbb{D}$, $\mathsf{App}(D)$ will be a finite, nonempty subset of $\mathbb{A}$. The space $\mathbb{A}$ can be canonically equipped with the disjoint union σ-algebra $\mathfrak{A} \coloneqq \bigoplus_{\hat{\varphi} \in \hat{\mathcal{G}}} \left(\{\hat{\varphi}\} \otimes \mathfrak{V}_{\hat{\varphi}}\right) \oplus \{\emptyset, \{(\square, \square)\}\}$.

Lemma 3.6.

(i) Let $\mathcal{A} \in \mathfrak{A}$. Then $\mathsf{App}^{-1}(\mathcal{A}) \coloneqq \{D \in \mathbb{D}\colon\ \mathsf{App}(D) \cap \mathcal{A} \neq \emptyset\}$ is a measurable subset of $\mathbb{D}$.

(ii) There exists a measurable function $\mathsf{app}\colon \mathbb{D} \to \mathbb{A}$ with the property that $\mathsf{app}(D) \in \mathsf{App}(D)$ for all $D \in \mathbb{D}$ (that is, app is a measurable selection of App).

We use measurable selections app of App to resolve in a measurable way nondeterminism occurring during the execution of an existential Datalog program. In a high-level view, suitable functions app determine the chase sequence (or, better yet "chase tree") for a program. This will become evident in the following sections.

3.4 Follow-Up Instances

It is crucial for our measurability considerations that there is a measurable correspondence between "intermediate instances" within the execution of the program and all the *follow-up instances* or *extensions* that emerge from such instances by a single rule application. Intuitively, whenever a rule is applicable (that is, its body is satisfied but its head is not), it may fire. If the rule is deterministic, then the fact from the head of the rule gets added to the current database instance. If the rule is probabilistic, then the fact from the head of the rule gets added with some valuation of the existentially quantified variable and we get a distribution over the follow-up instances according to the parameterized distribution from the original rule.

Let $\hat{\varphi}$ be a rule of the Datalog$^{\exists}$ version $\hat{\mathcal{G}}$ of $\mathcal{G}$. As in the previous section, $\mathbb{V}_{\hat{\varphi}}$ (with σ-algebra $\mathfrak{V}_{\hat{\varphi}}$) denotes the space of valuations of the free variables of $\hat{\varphi}$. If $\hat{\varphi}$ is existential, then $\mathbb{W}_{\hat{\varphi}}$ (with σ-algebra $\mathfrak{W}_{\hat{\varphi}}$ denotes the domain of the existentially quantified variale in $\hat{\varphi}$. That is, $\mathbb{W}_{\hat{\varphi}} = \mathbb{X}_{\psi}$ if ψ is the parameterized distribution of the rule φ of $\mathcal{G}$ from which $\hat{\varphi}$ was constructed. If $\hat{\varphi}$ is deterministic, then we let $\mathbb{W}_{\hat{\varphi}} = \{*\}$ be a fixed singleton set. This is just a technical device to unify the treatment of deterministic and existential rules later.

For every rule $\hat{\varphi}$ there is a function $f_{\hat{\varphi}}\colon \mathbb{V}_{\hat{\varphi}} \times \mathbb{W}_{\hat{\varphi}} \to \mathbb{F}_{\hat{\varphi}}$ mapping a valuation $\bar{a}$ and $b \in \mathbb{W}_{\hat{\varphi}}$ to the fact from the head of $\hat{\varphi}$ under $\bar{a}$:

- If $\hat{\varphi}$ is existential with free variables exactly $\bar{x}' = x_1, \dots, x_m$, rule head $\exists y\colon R_i(\bar{x}', y)$ and $\bar{a} = (a_1, \dots, a_m)$ is a valuation of $\bar{x}'$, then $f_{\hat{\varphi}}(\bar{a}, b) = R_i(\bar{x}', y)_{[x_1/a_1, \dots, x_m/a_m, y/b]}$.
- If $\hat{\varphi}$ is deterministic with rule head $R(\bar{x}')$ and $\bar{x}'$ and $\bar{a}$ exactly as above, then $f_{\hat{\varphi}}(\bar{a}, *) = R(\bar{x}')_{[x_1/a_1, \dots, x_m/a_m]}$.

It is easy to see that $f_{\hat{\varphi}}$ is measurable for all $\hat{\varphi}$. (A formal proof is contained in the full version of this paper [20].)

Definition 3.7. We define two *extension functions*:

- For $\mathbb{U} \coloneqq \{(D, \hat{\varphi}, \bar{a}, b)\colon D \in \mathbb{D}, \hat{\varphi} \in \hat{\mathcal{G}}, \bar{a} \in \mathbb{V}_{\hat{\varphi}} \text{ and } b \in \mathbb{W}_{\hat{\varphi}}\}$, we let $\mathsf{ext}\colon \mathbb{U} \to \mathbb{D}$ be the function defined via

$$\mathsf{ext}(D, \hat{\varphi}, \bar{a}, b) \mapsto D \cup \{f_{\hat{\varphi}}(\bar{a}, b)\}. \tag{3.C}$$

[1] We assumed here that all the positions where a variable x occurs are typed equally, i. e. have the same domain. In general, we only require that the domains of x are Polish subspaces of a common larger Polish space and if x occurs in the head of the rule, this larger space is the domain of the corresponding attribute in the head relation.

- Suppose $\hat{\mathcal{G}} = \{\hat{\varphi}_1, \dots, \hat{\varphi}_k\}$ and let $\vec{\ell} = (\ell_1, \dots, \ell_k)$ be any k-tuple of non-negative integers. Let $\mathbb{U}_{\vec{\ell}}$ e the set of tuples

  $$(D, \hat{\varphi}_1, \bar{a}_{11}, b_{11}, \dots, \hat{\varphi}_1, \bar{a}_{1\ell_1}, b_{1\ell_1}, \dots, \hat{\varphi}_k, \bar{a}_{k\ell_k}, b_{k\ell_k})$$

  where $D \in \mathbb{D}$, $a_{ij} \in \mathbb{V}_{\hat{\varphi}_i}$ and $b_{ij} \in \mathbb{W}_{\hat{\varphi}_i}$. Then $\mathsf{Ext}_{\vec{\ell}} \colon \mathbb{U}_{\vec{\ell}} \to \mathbb{D}$ is the function defined via

  $$\begin{aligned} &\mathsf{Ext}_{\vec{\ell}}(D, \hat{\varphi}_1, \bar{a}_{11}, b_{11}, \dots, \hat{\varphi}_k, \bar{a}_{k\ell_k}, b_{k\ell_k}) \\ &= D \cup \bigcup_{\substack{1 \le i \le k \\ 1 \le j_i \le \ell_i}} \{f_{\hat{\varphi}_i}(\bar{a}_{ij_i}, b_{ij_i})\}. \end{aligned} \tag{3.D}$$

$\mathbb{U}$ is the disjoint union of the spaces $\mathbb{D} \times \{\hat{\varphi}\} \times \mathbb{V}_{\hat{\varphi}} \times \mathbb{W}_{\hat{\varphi}}$ for $\hat{\varphi} \in \hat{\mathcal{G}}$ and is as such equipped with a σ-algebra $\mathfrak{U}$ in the straightforward way. Similarly, $\mathbb{U}_{\vec{\ell}}$ is the product space $\mathbb{D} \times \prod_{\hat{\varphi} \in \hat{\mathcal{G}}} (\{\hat{\varphi}\} \times \mathbb{V}_{\hat{\varphi}} \times \mathbb{W}_{\hat{\varphi}})^{\ell_i}$ and accordingly equipped with a product σ-algebra $\mathfrak{U}_{\vec{\ell}}$.

As the following lemma states, the functions we just defined are measurable. The proof of this result can be found in the full version of this paper [20].

Lemma 3.8.

(i) The function ext *is* $(\mathfrak{U}, \mathfrak{D})$*-measurable.*

(ii) The function $\mathsf{Ext}_{\vec{\ell}}$ *is* $(\mathfrak{U}_{\vec{\ell}}, \mathfrak{D})$*-measurable for all* $\vec{\ell} \in \mathbb{N}^k$.

For the following, we let ξ and $\Xi_{\vec{\ell}}$ denote the characteristic functions of the graphs of ext and $\mathsf{Ext}_{\vec{\ell}}$, respectively. That is, $\xi \colon \mathbb{U} \to \{0, 1\}$ is defined by

$$\xi(D, \hat{\varphi}, \bar{a}, b, D') = \begin{cases} 1 & \text{if } \mathsf{ext}(D, \hat{\varphi}, \bar{a}, b) = D' \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{3.E}$$

$\Xi_{\vec{\ell}}$ is defined similarly. (Cf. Definition 3.7 for the definition of $\mathbb{U}$.)

Corollary 3.9.

(i) The function ξ *is* $(\mathfrak{U} \otimes \mathfrak{D}, \mathcal{B}or(\mathbb{R}))$*-measurable.*

(ii) The function $\Xi_{\vec{\ell}}$ *is* $(\mathfrak{U} \otimes \mathfrak{D}, \mathcal{B}or(\mathbb{R}))$*-measurable for all* $\vec{\ell} \in \mathbb{N}^k$.

Proof. By Lemma 3.8, the functions ext and $\mathsf{Ext}_{\vec{\ell}}$ are measurable. Thus, their graphs are measurable sets in the corresponding product space. Since characteristic functions of measurable sets are measurable, the claim follows. □

3.5 Induced Functional Dependencies

With every existential rule $\hat{\varphi}$ of $\hat{\mathcal{G}}$, we associate a *functional dependency* $\mathsf{FD}(\hat{\varphi})$ in the following way. Suppose R_i is the relation in the head of $\hat{\varphi}$ with attributes $A_1, \dots, A_k$. Then $\mathsf{FD}(\hat{\varphi})$ is the functional dependency $R_i \colon A_1, \dots, A_{k-1} \to A_k$. That is, the functional dependency expresses that there is at most one value of the random (resp. existential) attribute when all other attribute values are fixed, cf. [3, p. 22:8]. The following is then easy to check using the definitions of App, $f_{\hat{\varphi}}$ and ext.

Lemma 3.10 (cf. [3, Proposition 4.2])**.** *Let* $\hat{\varphi}$ *be an existential rule of* $\hat{\mathcal{G}}$*. Then the following holds:*

(i) Every database instance D *of schema* $\mathcal{E}$ *satisfies* $\mathsf{FD}(\hat{\varphi})$.

(ii) If D *is a database instance and* $(\hat{\varphi}, \bar{a}) \in \mathsf{App}(D)$ *for some* $\bar{a}$*, then* D *satisfies* $\mathsf{FD}(\hat{\varphi})$*. Moreover, for every* b*, the follow-up instance* $\mathsf{ext}(D, \hat{\varphi}, \bar{a}, b)$ *satisfies* $\mathsf{FD}(\hat{\varphi})$ *as well.*

4 SEQUENTIAL PROBABILISTIC CHASE

As in [3], the chase of a GDatalog program $\mathcal{G}$ corresponds to chasing its Datalog$^{\exists}$ version $\hat{\mathcal{G}}$. The authors of [3] thus construct a "chase tree" for the existential Datalog program $\hat{\mathcal{G}}$ whose nodes are labeled with database instances and whose edges are derived from rule applications. The edges are labeled with the respective probability to go from the parent instance to its child by applying the rule. While we follow the general spirit of this approach, it is due to the involvement of uncountable domains no longer sufficient to label the edges this way. Instead, we label nodes with the probability distribution that is induced by the application of the rule. This section is devoted to formally describe the above and to demonstrate how such chase trees induce a stochastic process.

From a measure-theoretic point of view, there is no need to associate the execution of Datalog program to a tree in the way we are going to do it. We believe though, that doing so is beneficial for exposing the intuition behind the underlying stochastic process and for emphasizing the connections to the original approach in [3].

4.1 Chase Steps and Chase Trees

A *chase step* captures the semantics of applying a single (applicable) rule in an input instance. (The reader may want to compare this with the discrete version in [3, p. 22:12].)

Definition 4.1 (Chase Step)**.** A tuple $(D, \hat{\varphi}, \bar{a}, \mathcal{D}, \mu)$ is called a *(sequential) chase step* for $\mathcal{G}$ if

- D is a database instance,
- $(\hat{\varphi}, \bar{a}) \in \mathsf{App}(D)$, provided $\mathsf{App}(D) \neq \{(\square, \square)\}$,
- $\mathcal{D} = \mathsf{ext}(D, \hat{\varphi}, \bar{a}, \mathbb{W}_{\hat{\varphi}}) = \bigcup_{b \in \mathbb{W}_{\hat{\varphi}}} \mathsf{ext}(D, \hat{\varphi}, \bar{a}, b)$, and
- μ is the probability measure on the trace σ-algebra $\mathfrak{D} \restriction \mathcal{D}$ on $\mathcal{D}$ where for all measurable $\mathcal{E} \subseteq \mathcal{D}$, in the case of $\hat{\varphi}$ being an existential rule of $\hat{\mathcal{G}}$,

  $$\mu(\mathcal{E}) = \int_{\mathbb{W}_{\hat{\varphi}}} \xi(D, \hat{\varphi}, \bar{a}, b, \mathcal{E}) \cdot \psi\langle\bar{a}\rangle(b) \; d\mu_\psi \tag{4.A}$$

  with ψ being the parameterized distribution from the rule of the original program $\mathcal{G}$ that generated the rule $\hat{\varphi}$ in $\hat{\mathcal{G}}$;
  and, in the case that $\hat{\varphi}$ is a non-existential rule and accordingly $\mathcal{D} = \{E\}$ for some $E \in \mathbb{D}$,

  $$\mu(\mathcal{E}) = \begin{cases} 1 & \text{if } \mathcal{E} = \{E\} \text{ and} \\ 0 & \text{if } \mathcal{E} = \emptyset. \end{cases} \tag{4.B}$$

We denote a chase step $(D, \hat{\varphi}, \bar{a}, \mathcal{D}, \mu)$ as $D \xrightarrow{\hat{\varphi}(\bar{a})} (\mathcal{D}, \mu)$ and say it *starts in* D*, uses* $\hat{\varphi}$ *with valuation* $\bar{a}$ and *goes into* $\mathcal{D}$ *with distribution* μ.

Note that the definition of μ in Eq. (4.B) can be seen as a special case of Eq. (4.A) (recall that for deterministic rules, $\mathbb{W}_{\hat{\varphi}}$ is a singleton set $\{*\}$, we may just let $\psi\langle\bar{a}\rangle(*) = \mu_\psi(*) = 1$ in this case). This allows us without loss of generality to uniformly treat all the chase step measures that appear later as if they were of shape (4.A).

Furthermore it follows that μ is indeed a probability measure on the trace σ-algebra $\mathbb{D} \restriction \mathcal{D}$. (A formal proof can be found in the full version of this paper [20].)

Given a database instance D, we can now argue about sequences of follow-up instances using sequences of chase steps. In this process however, sequences can branch, when the rules that are applied

are existential rules of the Datalog$^{\exists}$ version of $\mathcal{G}$. What we just described is formalized in the notion of *chase trees*. (See [3, p. 22:13] for a comparison with the discrete case.)

Definition 4.2 (Chase Tree)**.** Let D_0 be a database instance and let app be a measurable selection of App (which we call a *measurable chase sequence*). The *(sequential) chase tree* T_{app,D_0} for D_0 w. r. t. the program $\mathcal{G}$ and app is the labelled tree $T_{\mathsf{app},D_0} = (V, E, \Lambda)$ of countable depth with $\Lambda\colon v \in V \mapsto (D_v, \hat{\varphi}_v, \bar{a}_v, \mathcal{D}_v, \mu_v)$ such that

(i) For the root r of T_{app,D_0}, $D_r = D_0$.
(ii) For any node v, if $(\hat{\varphi}_v, \bar{a}_v) \neq (\square, \square)$, then the children of v are bijectively labelled with the instances from $\mathcal{D}_v$. Otherwise, v is a leaf.
(iii) For every node $v \in V$, the label $\Lambda(v)$ is either
- a (sequential) chase step $D_v \xrightarrow{\hat{\varphi}_v(\bar{a}_v)} (\mathcal{D}_v, \mu_v)$ where $(\hat{\varphi}_v, \bar{a}_v) = \mathsf{app}(D_v)$, or
- a tuple $(D_v, \square, \square, \{D_v\}, D_v \mapsto 1)$ with $\mathsf{app}(D_v) = (\square, \square)$.

D_0 is also called the *root instance* of T_{app,D_0}.

It is easy to see that T_{app,D_0} is indeed uniquely determined by app and D_0. Formally, this can be shown by induction over its levels.

Note that if the rules of the program $\mathcal{G}$ only use discrete distributions, then any function app of fitting domain and range is measurable. Moreover, modulo our changes to the definition of the existential program we compile from $\mathcal{G}$, we obtain the same chase trees as in [3] with just a more complicated labelling (that would be unnecessarily complicated for a completely discrete setting).

4.2 Mapping Paths to Instances

We aim to "project" paths in a chase tree T_{app,D_0} to the collections of facts they represent, that is, the limit their instance labels approach with respect to set union. In the case of finite paths, this results in database instances. Infinite paths correspond to non-terminating chase sequences and to infinite collections of facts. As we always require database instances to be finite, these infinite paths are *not* database instances, and we will treat them as error events.

This section contains some auxiliary measurability results, whose proofs can be found in the full version of this paper [20].

For measurable chase sequence app, let $\vdash_{\mathsf{app}}$ be the relation on $(\mathbb{D}^2, \mathfrak{D}^{\otimes 2})$ with $D \vdash_{\mathsf{app}} D'$ if and only if

- $D \xrightarrow{\hat{\varphi}(\bar{a})} (\mathcal{D}, \mu)$ for some $\mathcal{D} \ni D'$ where $(\hat{\varphi}, \bar{a}) = \mathsf{app}(D)$; or
- $D = D'$ and $\mathsf{app}(D) = (\square, \square)$.

Lemma 4.3. *Seen as a set in* $\mathbb{D}^2$, $\vdash_{\mathsf{app}} \in \mathfrak{D}^{\otimes 2}$.

Define sets

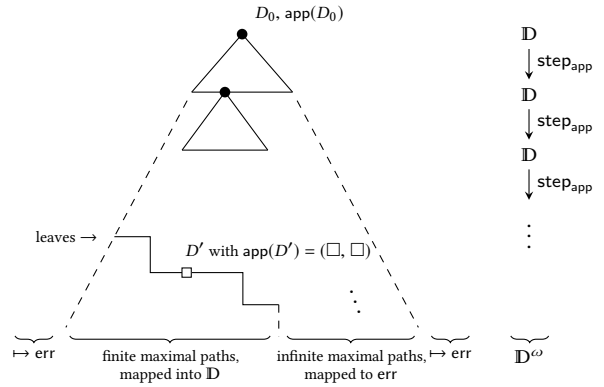
$$\mathsf{paths}(\mathsf{app}) \coloneqq \{(D_0, D_1, \dots) \in \mathbb{D}^\omega \colon D_i \vdash_{\mathsf{app}} D_{i+1} \text{ for all } i \in \mathbb{N}\}$$

$$\mathsf{paths}(\mathsf{app}, D_0) \coloneqq \mathsf{paths}(\mathsf{app}) \cap \left(\{D_0\} \times \mathbb{D}^\omega\right) \quad \text{f. a. } D_0 \in \mathbb{D}.$$

Note that $\mathsf{paths}(\mathsf{app}, D_0)$ is the set of paths of the tree T_{app,D_0} (where finite maximal sequences have been extended to infinite sequences by repeating the label of the leaf node). The following is a direct consequence of Lemma 4.3.

Corollary 4.4. *It holds that* $\mathsf{paths}(\mathsf{app}), \mathsf{paths}(\mathsf{app}, D_0) \in \mathfrak{D}^{\otimes\omega}$ *for all measurable chase sequences* app *and all fixed* $D_0 \in \mathbb{D}$.

Figure 1: Paths of the Sequential Chase Tree



Call a sequence $\vec{D} = (D_0, D_1, \dots) \in \mathbb{D}^\omega$ *stable at* i if $\vec{D} \in \mathbb{D}^{i-1} \times \mathsf{diag}(\mathbb{D}^\omega)$. If there exists $i \in \mathbb{N}$ such that $\vec{D}$ is stable at i, then we call $\vec{D}$ *stable*. Note that $\mathsf{diag}(\mathbb{D}^\omega) \in \mathfrak{D}^{\otimes\omega}$ since $(\mathbb{D}, \mathfrak{D})$ is standard Borel (Theorem A.1).

We now define a function $\mathsf{lim\text{-}inst}_{\mathsf{app}}$ that maps paths in $\mathbb{D}^\omega$ to their associated instance (or the error event "err"):

$$\mathsf{lim\text{-}inst}_{\mathsf{app}}(\vec{D}) \coloneqq \begin{cases} D_i & \text{if } \vec{D} \in \mathsf{paths}(\mathsf{app}) \text{ and } \vec{D} \text{ is stable at } i \\ \mathrm{err} & \text{otherwise.} \end{cases} \tag{4.C}$$

Similarly, for all $D_0 \in \mathbb{D}$, we define $\mathsf{lim\text{-}inst}_{\mathsf{app},D_0}\colon \mathbb{D}^\omega \to \mathbb{D}_{\mathrm{err}}$ using $\mathsf{paths}(\mathsf{app}, D_0)$ instead of $\mathsf{paths}(\mathsf{app})$ in Eq. (4.C).

Lemma 4.5. *Let* app *be a measurable chase sequence and let* D_0 *be a fixed database instance. Then:*

(i) The functions $\mathsf{lim\text{-}inst}_{\mathsf{app}}$ *and* $\mathsf{lim\text{-}inst}_{\mathsf{app},D_0}$ *are* $(\mathfrak{D}^{\otimes\omega}, \mathfrak{D}_{\mathrm{err}})$*-measurable.*
(ii) If $\vec{\mathcal{D}} \subseteq \mathsf{lim\text{-}inst}^{-1}_{\mathsf{app}}(\mathbb{D})$ *is measurable set in* $\mathfrak{D}^{\otimes\omega}$*, then its image* $\mathsf{lim\text{-}inst}_{\mathsf{app}}(\vec{\mathcal{D}})$ *is measurable in* $\mathfrak{D}$ *as well.*
Likewise, if $\vec{\mathcal{D}} \subseteq \mathsf{lim\text{-}inst}^{-1}_{\mathsf{app},D_0}(\mathbb{D})$ *is measurable in* $\mathfrak{D}^{\otimes\omega}$*, then its image* $\mathsf{lim\text{-}inst}_{\mathsf{app},D_0}(\vec{\mathcal{D}})$ *is measurable in* $\mathfrak{D}$.

4.3 Chase Trees as Markov Processes

In this subsection, we establish a correspondence between a chase tree for a given GDatalog program and a discrete-time Markov process whose state space is the (in general not countable) space of database instances. We have seen in the previous subsection how paths in a chase tree naturally correspond to a set of *paths* of such a process (w. r. t. the usual notion of path for stochastic processes) in the countably infinite product space $(\mathbb{D}^\omega, \mathfrak{D}^{\otimes\omega})$.

To obtain the correspondence to a Markov process, we need to show that the probabilistic transitions that are encoded within the nodes of any level of the chase tree, or, to be more precise, by its measurable chase sequence, describe a stochastic kernel from $(\mathbb{D}, \mathfrak{D})$ to itself.

The interpretation of the GDatalog semantics as a database-valued Markov process (which, by itself, was already recognized in [3, p. 22:14]) makes also apparent that a natural generalization

of the GDatalog language is to allow the input to be a (sub-)probabilistic database rather than a single instance. A GDatalog program then induces a mapping from a (sub-)probabilistic database to a sub-probabilistic database ("losing" the mass of "non-terminating" paths).

Let app be a measurable chase sequence. We define a function $\mathsf{step}_{\mathsf{app}} \colon \mathbb{D} \times \mathfrak{D} \to [0,1]$ as follows. Let $D \in \mathbb{D}$ and $\mathcal{E} \in \mathfrak{D}$.

- If $\mathsf{app}(D) = (\hat{\varphi}, \bar{a}) \neq (\square, \square)$ and $D \xrightarrow{\hat{\varphi}(\bar{a})} (\mathcal{D}, \mu)$ is the corresponding chase step, we let
$$\mathsf{step}_{\mathsf{app}}(D, \mathcal{E}) = \mu(\mathcal{D} \cap \mathcal{E}) = \int_{\mathbb{W}} \xi(D, \hat{\varphi}, \bar{a}, \cdot, \mathcal{D} \cap \mathcal{E}) \cdot \psi\langle \bar{a} \rangle \; d\mu_\psi$$
where $\mathbb{W} = \mathbb{W}_{\hat{\varphi}}$, ψ is the parameterized distribution of $\hat{\varphi}$ and ξ is the function from Eq. (3.E).
- Otherwise, if $\mathsf{app}(D) = (\square, \square)$, we let
$$\mathsf{step}_{\mathsf{app}}(D, \mathcal{E}) = \iota(D, \mathcal{E}) = \begin{cases} 1 & \text{if } D \in \mathcal{E} \text{ and} \\ 0 & \text{if } D \notin \mathcal{E}. \end{cases}$$
Recall that ι denotes the identity kernel.

The following proposition resolves the main technical obstacle for turning measurable chase sequences and sequential chase trees into Markov processes.

Proposition 4.6. *For all measurable chase sequences* app*, the step function* $\mathsf{step}_{\mathsf{app}}$ *is a stochastic kernel.*

Proof. Clearly, $\mathsf{step}_{\mathsf{app}}(D, \cdot)$ is a probability measure for all $D \in \mathbb{D}$. We need to show that $\mathsf{step}(\cdot, \mathcal{E})$ is $(\mathfrak{D}, \mathcal{B}or[0,1])$-measurable for all $\mathcal{E} \in \mathfrak{D}$. In order to show this, we fix $\mathcal{E} \in \mathfrak{D}$ and, moreover, demonstrate that the statement holds when we restrict ourselves to database instance D where a fixed rule $\hat{\varphi}$ fires according to app. That is, we concentrate on the restriction of the preimage of $\mathsf{step}_{\mathsf{app}}$ to $\mathbb{D}_{\hat{\varphi}} := \mathsf{app}^{-1}(\{\hat{\varphi}\} \times \mathbb{V}_{\hat{\varphi}})$. Therein, recall that $\mathbb{V}_{\hat{\varphi}}$ is the joint domain of the free variables of $\hat{\varphi}$.

(Note that the result clearly holds for the restriction of $\mathbb{D}$ to the set $\mathbb{D}_\square$ of instances with $\mathsf{app}(D) = (\square, \square)$ since on these instances, step is the identity kernel.)

Let us fix a rule $\hat{\varphi}$. Without restriction (see the short discussion below Definition 4.1), we may treat $\hat{\varphi}$ as if it were an existential rule. We let $\mathbb{V}_{\hat{\varphi}} =: \mathbb{V}$ and $\mathbb{W}_{\hat{\varphi}} =: \mathbb{W}$ for the space of the existentially quantified variable of $\hat{\varphi}$. Let ψ be the parameterized distribution occurring in the rule φ of $\mathcal{G}$ that $\hat{\varphi}$ originated from.

From Fact 2.3, we know that the following function is a stochastic kernel from $\mathbb{V}$ to $\mathbb{W}$:
$$G \colon \mathbb{V} \times \mathfrak{W} \to [0,1] \colon (\bar{a}, \mathcal{B}) \mapsto \int_{\mathcal{B}} \psi\langle \bar{a} \rangle \; d\mu_\psi.$$
Then the following function is a stochastic kernel as well:
$$K \colon (\mathbb{D}_{\hat{\varphi}} \times \mathbb{V}) \times \mathfrak{W} \to [0,1] \colon (D, \bar{a}, \mathcal{B}) \mapsto G(\bar{a}, \mathcal{B}).$$
Note that the function
$$\xi(\cdot, \hat{\varphi}, \cdot, \cdot, \mathcal{E}) \colon \mathbb{D}_{\hat{\varphi}} \times \mathbb{V} \times \mathbb{W} \to \{0,1\}$$
is measurable, as it is the characteristic function of the $\hat{\varphi}$-section of $\mathsf{ext}^{-1}(\mathcal{E})$, the latter of which is measurable by Lemma 3.8. Then (using Fact B.2 from Appendix B for $\xi(\cdot, \hat{\varphi}, \cdot, \cdot, \mathcal{E})$ and K), the following function is measurable:
$$K' \colon \mathbb{D}_{\hat{\varphi}} \times \mathbb{V} \to [0,1] \colon (D, \bar{a}) \mapsto \int_{\mathbb{W}} \xi(D, \hat{\varphi}, \bar{a}, \cdot, \mathcal{E}) \cdot \psi\langle \bar{a} \rangle \; d\mu_\psi.$$
Now observe that the function $h \colon \mathbb{D}_{\hat{\varphi}} \to \mathbb{D}_{\hat{\varphi}} \times \mathbb{V}$ with $h(D) = (\hat{\varphi}, \bar{a})$ is measurable due to the measurability of app (and using Fact B.1). Let "$<\alpha$" denote the interval $[0, \alpha) \subseteq [0,1]$. Then
$$h^{-1}(K'(<\alpha)) \in \mathfrak{D}. \tag{4.D}$$
Finally note that for all $D \in \mathbb{D}_{\hat{\varphi}}$, $b \in \mathbb{W}$ and $\bar{a}$ being the tuple from $\mathbb{V}$ with $\mathsf{app}(D) = (\hat{\varphi}, \bar{a})$ and $\mathcal{D}$ being the set of follow-up instances of D from the chase step, it holds that
$$\xi(D, \hat{\varphi}, \bar{a}, b, \mathcal{D} \cap \mathcal{E}) = \xi(D, \hat{\varphi}, \bar{a}, b, \mathcal{E}).$$
This means that $h^{-1}(K'(<\alpha)) = \mathsf{step}_{\mathsf{app}}(\cdot, \mathcal{E})^{-1}(<\alpha)$, so Eq. (4.D) shows the assertion. □

With Fact B.9, we obtain the following.

Corollary 4.7. *For every measurable chase sequence* app *and every initial (sub-)probability distribution p on* $(\mathbb{D}, \mathfrak{D})$*, there exists a Markov process with state space* $(\mathbb{D}, \mathfrak{D})$*, initial distribution* ν *and transition kernels* $\mathsf{step}_{\mathsf{app}}$*.*

Since every (sub-)probability distribution on $(\mathbb{D}^\omega, \mathfrak{D}^{\otimes\omega})$ (the path space of such a process) yields a push-forward (sub-)probability distribution on $(\mathbb{D}_{\mathsf{err}}, \mathfrak{D}_{\mathsf{err}})$ along $\mathsf{lim\text{-}inst}_{\mathsf{app}}$ (or $\mathsf{lim\text{-}inst}_{\mathsf{app}, D_0}$), every such Markov process defines an SubPDB.

Theorem 4.8. *For every measurable chase sequence* app *and all instances* $D_0 \in \mathbb{D}$*,* $\mathcal{G}$ *on input* D_0 *defines an SubPDB* $\Delta_{\mathsf{app}, D_0}$*.*

If Δ *is an SubPDB, then for every measurable chase sequence* app*,* $\mathcal{G}$ *(with input* Δ*) defines an SubPDB* Δ_{app}*.*

For the first part of the above theorem, we let the initial distribution of Corollary 4.7 be the Dirac one on the instance D_0. For the second part, the initial distribution is the (sub-)probability distribution of the input SubPDB.

Remark 4.9. In the end, we might want to get rid of the auxiliary relations that were created in the translation to the Datalog$^\exists$ program. This can be done in a measurable way by Fact 2.6, yielding again an SubPDB.

5 PARALLEL PROBABILISTIC CHASE

We obtain another variant of the chase procedure if we allow all applicable rules to fire simultaneously. The aim of this section is to formalize this notion of parallel chase for the GDatalog language and to subsequently exhibit how it relates to the sequential chase that we discussed in the previous section.

As most of the results can be obtained in a manner analogous to Section 4, we omit the details when they can be easily derived from ideas of the aforementioned section.

Throughout the following, we fix a GDatalog program $\mathcal{G}$ with its Datalog$^\exists$ version $\hat{\mathcal{G}}$ and assume that $\hat{\mathcal{G}} = \{\hat{\varphi}_1, \ldots, \hat{\varphi}_k\}$.

5.1 Parallel Chase Steps and the Parallel Tree

If D is a database instance, the *firing configuration* of D is the tuple $\vec{\ell}(D) = (\ell_1, \dots, \ell_k) \in \mathbb{N}^k$ where $\ell_i = |\{\bar{a} \colon (\hat{\varphi}_i, \bar{a}) \in \mathsf{App}(D)\}|$ for all $1 \le i \le k$. Note that the set $\mathbb{D}_{\vec{\ell}}$ of database instances having a fixed firing configuration $\vec{\ell}$ is measurable in $(\mathbb{D}, \mathfrak{D})$, since we know that App corresponds to an Relational Calculus view and the cardinalities in question can be obtained by a counting aggregation. This yields a measurable mapping by Fact 2.6.

Definition 5.1 (Parallel Chase Step)**.** A tuple $(D, A, \mathcal{D}, \mu)$ is called a *parallel chase step* for $\mathcal{G}$ if

- $D \in \mathbb{D}$ with some firing configuration $\vec{\ell} = (\ell_1, \dots, \ell_k)$,
- $A = \mathsf{App}(D)$ with $A \neq \{(\square, \square)\}$, say
$$A = \{(\hat{\varphi}_i, \bar{a}_{ij_i}) \colon 1 \le i \le k \text{ and } 1 \le j_i \le \ell_i\}$$
- with $\mathcal{Z}_{ij} = \{(\hat{\varphi}_i, \bar{a}_{ij}, b) \colon b \in \mathbb{W}_{\hat{\varphi}_i}\}$,
$$\mathcal{D} = \Xi_{\vec{\ell}}(D, \mathcal{Z}_{11}, \dots, \mathcal{Z}_{1\ell_1}, \dots, \mathcal{Z}_{k1}, \dots, \mathcal{Z}_{k\ell_k})$$
(Recall that $\Xi_{\vec{\ell}}$ is the characteristic function of the graph of $\mathsf{Ext}_{\vec{\ell}}$, cf. Definition 3.7 and Eq. (3.E).)
- μ is the probability measure on the trace σ-algebra of $\mathbb{D} \restriction \mathcal{D}$ with
$$\mu(\mathcal{E}) = \int_{\mathbb{W}} \Xi_{\vec{\ell}}\big(D, \hat{\varphi}_1, \bar{a}_{11}, b_{11}, \dots, \hat{\varphi}_k, \bar{a}_{k\ell_k}, b_{k\ell_k}, \mathcal{E}\big) \cdot \psi_1\langle \bar{a}_{11}\rangle(b_{11}) \cdots \psi_k\langle \bar{a}_{k\ell_k}\rangle(b_{k\ell_k}) \, d\Big(\mu_{\psi_1}^{\otimes \ell_1} \otimes \cdots \otimes \mu_{\psi_k}^{\otimes \ell_k}\Big)$$
where $\mathbb{W} = \mathbb{W}_{\hat{\varphi}_1}^{\ell_1} \times \cdots \times \mathbb{W}_{\hat{\varphi}_k}^{\ell_k}$ and ψ_i is the parameterized distribution of the rules $\hat{\varphi}_i$.

Just as in Section 4, we concentrate on the existential rules and interpret the deterministic ones as special cases. (See again the discussion below Definition 4.1.) In particular note that it poses no problem that in a database instance multiple deterministic rules with the same left-hand side might be applicable, by the definition of $\Xi_{\vec{\ell}}$. Also observe that μ from Definition 5.1 is again a well-defined probability measure.

We point out that in the definition of μ, we use the product density of the individual densities. We thus make an implicit independence assumption (cf. [25, Lemma 3.10]): all probabilistic rules firing in a parallel chase step sample their respective distributions independently. By Fubini's theorem Fact B.8 and the definition of $\Xi_{\vec{\ell}}$, the concrete order of the $(\hat{\varphi}_i, \bar{a}_{ij}, b_{ij})$ has no impact on μ whatsoever.

We denote parallel chase steps $(D, A, \mathcal{D}, \mu)$ as $D \xrightarrow{A} (\mathcal{D}, \mu)$. In the case of the sequential chase, there were multiple possible chase steps starting in a database instance D, depending on the choice of the selection app of App. This is no longer the case for the parallel chase. A parallel chase step in database instance D is (if it exists) unique.

Definition 5.2 (Parallel Chase Tree)**.** Let D_0 be a database instance. The *parallel chase tree* for D_0 w. r. t. the GDatalog program $\mathcal{G}$ is the labelled tree $T_{\mathsf{App}, D_0} = (V, E, \Lambda)$ of countable depth where $\Lambda \colon v \in V \mapsto (D_v, A_v, \mathcal{D}_v, \mu_v)$ such that

(i) For the root r of T_{App, D_0}, $D_r = D_0$.

(ii) For any node v, if $A_v \neq (\square, \square)$, then the children of v are bijectively labelled with the instances from $\mathcal{D}_v$. Otherwise, v is a leaf.

(iii) For every node $v \in V$, the label $\Lambda(v)$ is either

- a parallel chase step $D_v \xrightarrow{A_v} (\mathcal{D}_v, \mu_v)$ where $A_v = \mathsf{App}(D_v)$, or
- a tuple $(D_v, (\square, \square), \{D_v\}, \delta_{D_v})$ with $\mathsf{App}(D_v) = \{(\square, \square)\}$.

D_0 is called the root instance of T_{App, D_0}.

As in Section 4, it is easy to see that T_{App, D_0} determined by D_0 (which can be shown inductively over the levels of the tree). The key difference between Definition 4.2 and Definition 5.2 that parallel chase steps attain the role that sequential chase steps had in sequential chase trees in Item (iii).

Based on the ideas of Section 4.2 we derive functions $\vdash_{\mathsf{App}}$, $\mathsf{lim\text{-}inst}_{\mathsf{App}}$ and $\mathsf{lim\text{-}inst}_{\mathsf{App}, D_0}$ (for all $D_0 \in \mathbb{D}$) as well as sets $\mathsf{paths}(\mathsf{App})$ and $\mathsf{paths}(\mathsf{App}, D_0)$ (for all $D_0 \in \mathbb{D}$). In a similar manner as is done there, they enjoy the properties from Lemmas 4.3 and 4.5 and Corollary 4.4. The detailed definitions and arguments can be found in the full version of this paper [20].

5.2 The Markov Process for Parallel Chasing

In analogy to Section 4.3, we show in this section how the parallel chase defines a Markov process of database instances. Throughout this section, let $\mathcal{G}$ again be a fixed GDatalog program.

We define a function $\mathsf{step}_{\mathsf{App}} \colon \mathbb{D} \times \mathfrak{D} \to [0, 1]$ as follows. Let $D \in \mathbb{D}$ and $\mathcal{E} \in \mathfrak{D}$.

- If $\mathsf{App}(D) \neq \{(\square, \square)\}$ and $D \xrightarrow{A} (\mathcal{D}, \mu)$ is the corresponding parallel chase step, we let $\mathsf{step}_{\mathsf{App}}(D, \mathcal{E}) = \mu(\mathcal{D} \cap \mathcal{E})$ with μ as in Definition 5.1.
- If $\mathsf{App}(D) = \{(\square, \square)\}$, we let $\mathsf{step}_{\mathsf{App}}(D, \mathcal{E}) = \iota(D, \mathcal{E})$ where ι is the identity kernel.

Proposition 5.3. *The parallel step function* $\mathsf{step}_{\mathsf{App}}$ *is a stochastic kernel.*

This works similar to the proof of Proposition 4.6 using Fubini's theorem (Fact B.8) to break up the product measure and handling the resulting integrals iteratively. The proof can be found in full version of this paper [20]. As in Section 4.3 we eventually obtain:

Corollary 5.4. *For every intial (sub-)probability distribution* p *on* $(\mathbb{D}, \mathfrak{D})$*, there exists a Markov process with state space* $(\mathbb{D}, \mathfrak{D})$*, initial distribution* p *and transition kernels* $\mathsf{step}_{\mathsf{App}}$*.*

Theorem 5.5. *For all instances* $D_0 \in \mathbb{D}$*, program* $\mathcal{G}$ *with input* D_0 *defines an SubPDB* $\Delta_{\mathsf{App}, D_0}$*. If* Δ *is an SubPDB, then* $\mathcal{G}$ *(with input* Δ*) defines an SubPDB* Δ_{App}*.*

6 SEMANTIC PROPERTIES OF GDATALOG

6.1 Chase Independence

Let $\mathcal{G}$ be a GDatalog program. From Theorems 4.8 and 5.5 we know that $\mathcal{G}$, given some input, produces SubPDBs. Let $D_0 \in \mathbb{D}$ and fix a measurable chase sequence app.

Theorem 6.1. $\Delta_{\mathsf{app}, D_0} = \Delta_{\mathsf{App}, D_0}$.

Theorem 6.1 means that no matter which chase sequence we use (or whether we use the parallel chase), the resulting SubPDB is the

same, rendering our semantics fairly robust. The proof can be found in the full version of the paper. The basic idea is to partition events in the resulting measurable space into classes based on how they were produced in T_{app,D_0} and T_{App,D_0}, respectively. This includes fixing a chase sequence for the sequential chase, a sequence of firing configurations for the parallel chase and a correspondence between applicable pairs in both of the chase procedures. Then it may be argued how the resulting integrals may be replaced and then, using Fubini's theorem, rearranged to obtain Theorem 6.1. This is another non-trivial step that requires reasoning about which replacements are allowed with respect to rule applicability.

Theorem 6.2. *If Δ is an SubPDB, then $\mathcal{G}$ (with input Δ) defines the same output SubPDB Δ' regardless of the chase procedure that is used.*

6.2 Remarks Regarding the Original Semantics

In this subsection, we want to briefly discuss how our semantics relate to the one proposed by Bárány et al. [3]. As mentioned in the introduction, Bárány et al. wanted their probabilistic Datalog to be invariant under first-order equivalence. For example, they want the program $\mathcal{G}_0$ from Example 1.1 to be equivalent to the program $\mathcal{G}_0''$ consisting of a single rule $R(\mathsf{Flip}\langle 1/2\rangle) \leftarrow \top$. Note that $\mathcal{G}_0$ and $\mathcal{G}_0''$ are not equivalent under our semantics.

The way Bárány et al. achieved this behaviour is by allowing each probability distribution to only be sampled once with each parameter value. So where we say that each probabilistic rule samples at most once, they tie samples to the (name of) the distribution. This is why renaming a probability distribution may change the semantic behaviour of a program, as illustrated by the program $\mathcal{G}_0'$ in Example 1.1. Of course it may sometimes be desirable to sample several times from the same distribution with the same parameters. Bárány et al. deal with this by "tagging" individual applications with additional parameters instantiated by constants.

To illustrate how we can simulate the behavior of Bárány et al.'s semantics with our version, consider the following program:

$$\begin{array}{|rcl|}\hline \mathcal{H}\colon\ R(\mathsf{Flip}\langle 1/2\rangle) & \leftarrow & \top \\ S(\mathsf{Flip}\langle 1/2\rangle) & \leftarrow & \top \\ \hline \end{array}$$

With Bárány et al.'s semantics, $\mathcal{H}$ has outcomes $\{R(0), S(0)\}$ and $\{R(1), S(1)\}$ with probability $1/2$ each. With our semantics, it has out comes $\{R(0), S(0)\}$, $\{R(1), S(0)\}$, $\{R(0), S(1)\}$, and $\{R(1), S(1)\}$ with probability $1/4$ each. However, we can easily simulate $\mathcal{H}$ under Bárány et al.'s semantics by pulling out the sampling to a separate rule, as in the following program:

$$\begin{array}{|rcl|}\hline \mathcal{H}'\colon\ A(\mathsf{Flip}\langle 1/2\rangle) & \leftarrow & \top \\ R(x) & \leftarrow & A(x) \\ S(x) & \leftarrow & A(x) \\ \hline \end{array}$$

This program has outcomes $\{R(0), S(0), A(0)\}$ and $\{R(1), S(1), A(1)\}$ with probability $1/2$ each. We can ignore the auxiliary predicate A and restrict the resulting probabilistic database to the schema $\{R, S\}$ without changing the probabilities.

This simple argument can be generalized to arbitrary programs. We leave the details to the reader. Let us remark that it is similarly easy to simulate our semantics with that of Bárány et al. All our results would also hold starting from Bárány et al.'s semantics for discrete distributions, so it is really just a matter of taste which version the reader prefers; technically, it makes no difference.

6.3 Termination Behavior

As we have seen in Section 4 resp. Section 5, the execution of a GDatalog program corresponds to a Markov process. Every point in the ith level of the path space $(\mathbb{D}^\omega, \mathfrak{D}^{\otimes\omega})$ can be seen as corresponding to a program configuration and every path in $(\mathbb{D}^\omega, \mathfrak{D}^{\otimes\omega})$ as a program run. A run is called *terminating* if it corresponds to a finite path in the respective chase tree. The program $\mathcal{G}$ is called *terminating* if all runs terminate and is called *almost surely terminating (AST)* if the set of non-terminating runs is a set of measure zero. The following result from the original paper trivially extends to our setting, with the notion of *weak acyclicity* remaining unchanged (see [3]).

Theorem 6.3 (cf. [3, Theorem 3.10]). *Let $\mathcal{G}$ be a GDatalog program. If $\mathcal{G}$ is weakly acyclic, then $\mathcal{G}$ terminates.*

Basically it states that whenever there are no circular dependencies involving probabilistic rules, then all paths in any chase tree are finite. Note that in our probabilistic setting, the mere existence of infinite paths is no problem, as long as they are an event of probability zero (or, perhaps, of sufficiently low probability).

One might wonder now, whether there are more sophisticated criteria that ensure *almost sure* termination of $\mathcal{G}$, that is, termination with probability 1. We informally argue that in the presence of circular dependencies involving continuous distributions, we cannot hope for a more powerful criterion other than imposing some acyclicity constraint with respect to the probabilistic rules. Recall that an instance corresponds to a leaf node of a chase tree, if no rule is applicable anymore. For the existential rules of Datalog$^\exists$, this is in particular the case, whenever its parameterized distribution has been sampled before along with the same valuation. The circular dependency means that, one way or another, the sample result is at some point fed into the same rule again as a parameter. That is, a probabilistic rule application attributes to the termination of $\mathcal{G}$, if it samples from some "good set" that is present in the current intermediate instance. Since intermediate instances are always finite collections of facts and since continuous distributions typically allot measure zero to every countable subset of the sampling space, the probability to generate "bad samples" will always be 1, so the program will almost surely not terminate.

In cycles that only include discrete probability distributions, we see some hope though to bound the probability of terminating in terms of the parameterized distribution that is used. We are currently investigating this and refer to future work.

7 CONCLUSION

In this paper, we generalise the probabilistic Datalog language introduced in [3] to a setting that allows for continuous probability distributions. Such distributions, for example, normal or exponential distributions, appear naturally in many application scenarios, and indeed Bárány et al. [3] explicitly asked for a continuous generaliztation of their language.

To summarize the technical developments laid out in this paper, let us give a high-level view of this semantics. A probabilistic

Datalog program can sample from probability distributions to generate values appearing in its intensional facts, and it can do so recursively. The semantics is then described via infinite chase trees, where different branches correspond to different samples. While in the discrete case it is relatively straightforward to formalise this, with continuous distributions, where a node in the chase tree may have unconutably many children (corresponding to the outcomes of the application of a single probabilistic rule), this becomes technically challenging. Using advanced tools from probability theory, we show that such an uncountable chase tree can be viewed as a Markov process (a fairly well-behaved stochastic process). Associated with each Markov process is a probability measure on its paths, that is, the paths of the chase tree. Each path of the tree corresponds to a possibly infinite collection of facts generated along the path, and, taking into account that database instances are required to be finite and thus only finite paths correspond to instances, we can "project" the probability measure on the paths back to instances and thereby obtain the PDB "generated" by the Datalog program.

Future Work. The second part of Probabilistic Programming Datalog of [3] allows the user to condition the result of the generative part based on logical constraints. However, in the continuous setting this causes some delicate issues. For example, it might be reasonable to use constraints involving equality, yet, for example equality on $\mathbb{R}$ is under the hood the diagonal in $\mathbb{R}^2$ which is a set of Lebesgue measure zero. Work from the area of probabilistic programming [5] suggests that completing GDatalog towards a generalized Probabilistic Programming Datalog is a nontrivial ask. In particular, conditioning on events of measure zero can yield paradoxical results (see Borel-Kolmogorov paradox [28, p. 50 et seq.])

We are currently investigating the termination behavior of GDatalog programs that are not weakly acyclic but contain only discrete distributions (cf. Section 6.3).

As GDatalog progams essentially produce SubPDBs, it seems natural to ask how powerful they are as a representation system for (infinite) PDBs. (Bárány et al. showed that PPDL is a complete representation system for finite PDBs.)

ACKNOWLEDGMENTS

We are grateful to Benny Kimelfeld for bringing our attention to GDatalog and to Marcel Hark for discussions regarding termination.

This work is supported by the German Research Foundation (DFG) under grants GR 1492/16-1 and GRK 2236 (UnRAVeL).

REFERENCES

[1] Charu C. Aggarwal and Philip S. Yu. A Survey of Uncertain Data Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 21(5):609–623, 2009. doi:10.1109/TKDE.2008.190.

[2] Parag Agrawal and Jennifer Widom. Continuous Uncertainty in Trio. In *Proceedings of the 3rd VLDB workshop on Management of Uncertain Data (MUD 2009)*, pages 17–32, Enschede, The Netherlands, 2009. Centre for Telematics and Information Technology (CTIT).

[3] Vince Bárány, Balder ten Cate, Benny Kimelfeld, Dan Olteanu, and Zografoula Vagena. Declarative Probabilistic Programming with Datalog. *ACM Transactions on Database Systems (TODS)*, 42(4):22:1–22:35, 2017. doi:10.1145/3132700.

[4] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019.

[5] Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Van Gael. Measure Transformer Semantics for Bayesian Machine Learning. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:11)2013.

[6] Zhuhua Cai, Zografoula Vagena, Luis Perez, Subramanian Arumugam, Peter J. Haas, and Christopher Jermaine. Simulation of Database-Valued Markov Chains using SimSQL. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)*, pages 637–648, New York, NY, USA, 2013. ACM. doi:10.1145/2463676.2465283.

[7] Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 76(1), 2017.

[8] D. J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes, Volume II: General Theory and Structure*. Probability and its Applications. Springer, New York, NY, USA, 2nd edition, 2008. doi:10.1007/978-0-387-49835-5.

[9] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[10] Luc De De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*, volume 10. Morgan & Claypool Publishers, 2016. doi:10.2200/S00692ED1V01Y201601AIM032.

[11] Daniel Deutch, Christoph Koch, and Tova Milo. On Probabilistic Fixpoint and Markov Chain Query Languages. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2010)*, pages 215–226, New York, NY, USA, 2010. ACM. doi:10.1145/1807085.1807114.

[12] David H. Fremlin. *Measure Theory. Vol. II: Broad Foundations*. Torres Fremlin, 2nd edition, 2010.

[13] David H. Fremlin. *Measure Theory. Vol. IV: Topological Measure Spaces, Part I*. Torres Fremlin, 2nd edition, 2013.

[14] Norbert Fuhr. Probabilistic Datalog – A Logic for Powerful Retrieval Methods. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1995)*, pages 282–290, New York, NY, USA, 1995. ACM. doi:10.1145/215206.215372.

[15] Marie Gaudard and Donald Hadwin. Sigma-Algebras on Spaces of Probability Measures. *Scandinavian Journal of Statistics*, 16(2):169–175, 1989.

[16] Noah D. Goodman. The principles and practice of probabilistic programming. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2013)*, pages 399–402, New York, NY, USA, 2013. ACM. doi:10.1145/2429069.2429117.

[17] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A Language for Generative Models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI 2008)*, pages 220–229, Arlington, VA, USA, 2008. AUAI Press.

[18] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering (FOSE 2014)*, pages 167–181, New York, NY, USA, 2014. ACM. doi:10.1145/2593882.2593900.

[19] Georg Gottlob, Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I. Simari. Query Answering under Probabilistic Uncertainty in Datalog +/- Ontologies. *Annals of Mathematics and Artificial Intelligence*, 69(1):37–72, 2013. doi:10.1007/s10472-013-9342-1.

[20] Martin Grohe, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Peter Lindner. Generative Datalog with Continuous Distributions. *arXiv e-prints*, 2020. URL: https://arxiv.org/abs/2001.06358.

[21] Martin Grohe and Peter Lindner. Probabilistic Databases with an Infinite Open-World Assumption. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2019)*, pages 17–31, New York, NY, USA, 2019. ACM. doi:10.1145/3294052.3319681.

[22] Martin Grohe and Peter Lindner. Infinite Probabilistic Databases. In Carsten Lutz and Jean Christoph Jung, editors, *23rd International Conference on Database Theory (ICDT 2020)*, volume 155 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2020/11940, doi:10.4230/LIPIcs.ICDT.2020.16.

[23] Bernd Gutmann, Manfred Jaeger, and Luc De Raedt. Extending ProbLog with Continuous Distributions. In *Inductive Logic Progamming (ILP 2010)*, volume 6489 of *Lecture Notes in Computer Science*, pages 76–91, Berlin, Germany and Heidelberg, Germany, 2011. Springer. doi:10.1007/978-3-642-21295-6_12.

[24] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Perez, Chris Jermaine, and Peter 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. doi:10.1145/2000824.2000828.

[25] Olav Kallenberg. *Foundations of Modern Probability*. Springer Series in Statistics. Probability and its Applications. Springer, New York, NY, USA, 2nd edition, 2002. doi:10.1007/978-1-4757-4015-8.

[26] Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A Short Introduction to Probabilistic Soft Logic. In *Proceedings of the*

NIPS Workshop on Probabilistic Programming: Foundations and Applications, pages 1–4, 2012.

[27] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA, 2009.

[28] Andrei Nikolajewitsch Kolmogorov. *Foundations of the Theory of Probability*. Chelsea Publishing Company, New York, NY, USA, 2nd English edition, 1956.

[29] C. Kuratowski and C. Ryll-Nardzewski. A General Theorem on Selectors. *Bulletin of the Polish Academy of Sciences*, 13:397–403, 1965.

[30] Eckhard Limpert, Werner A. Stahel, and Markus Abbt. Log-normal Distributions across the Sciences: Keys and Clues. *BioScience*, 51(5):341–352, 2001. doi:10.1641/0006-3568(2001)051[0341:LNDATS]2.0.CO;2.

[31] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, David L. Ong, and Andrey Kolobov. BLOG: Probabilistic Models with Unknown Objects. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 1352–1359, San Francisco, CA, USA, 2005. Morgan Kaufmann.

[32] Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. R2: An Efficient MCMC Sampler for Probabilistic Programs. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2476–2482. AAAI Press, 2014.

[33] Avi Pfeffer. Figaro: An Object-Oriented Probabilistic Programming Language. Technical report, Charles River Analytics, 2009.

[34] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1–2):107–136, 2006. doi:10.1007/s10994-006-5833-1.

[35] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne Hambrusch, and Rahul Shah. Orion 2.0: Native Support for Uncertain Data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD 2008)*, pages 1239–1242, New York, NY, USA, 2008. ACM. doi:10.1145/1376616.1376744.

[36] Parag Singla and Pedro Domingos. Markov logic in infinite domains. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI 2007)*, pages 368–375, Arlington, VA, USA, 2007. AUAI Press.

[37] Sashi Mohan Srivastava. *A Course on Borel Sets*, volume 180 of *Graduate Texts in Mathematics*. Springer, New York, NY, USA, 1998.

[38] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool, San Rafael, CA, USA, 1st edition, 2011. doi:10.2200/S00362ED1V01Y201105DTM016.

[39] David Tolpin, Jan-Willem van de Meent, and Frank Wood. Probabilistic Programming in Anglican. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2015)*, volume 9286 of *Lecture Notes in Computer Science*, pages 308–311, Cham, Switzerland, 2015. Springer International Publishing. doi:10.1007/978-3-319-23461-8_36.

[40] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming. *arXiv e-prints*, 2018. URL: https://arxiv.org/abs/1809.10756.

[41] Brend Wanders, Maurice van Keulen, and Jan Flokstra. JudgeD: A Probabilistic Datalog with Dependencies. In *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence: Technical Reports WS-16-01 – WS-16-15*, Palo Alto, CA, USA, 2016. AAAI Press.

[42] Yi Wu, Siddharth Srivastava, Nicholas Hay, Simon Du, and Stuart Russell. Discrete-Continuous Mixtures in Probabilistic Programming: Generalized Semantics and Inference Algorithms. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, volume 80 of *Proceedings of Machine Learning Research*, pages 5343–5352. PMLR, 2018.

A STANDARD PROBABILISTIC DATABBASES

All facts we use about standard probabilistic databases we use in this paper were shown in [22] with the exception of the following.

Theorem A.1. *The instance measurable space of any standard PDB* $(\mathbb{D}, \mathfrak{D})$ *is standard Borel, as is its restriction* $(\mathbb{D}_*, \mathfrak{D}_*)$ *to set instances.*

Proof Sketch. This is an instantiation of a known result from point process theory and the theory of random measures. We use the notation from [8]. For any standard Borel space $(\mathbb{X}, \mathfrak{X})$, the set $\mathcal{N}_{\mathbb{X}}^{\#}$ of $\mathbb{N} \cup \{\infty\}$-valued measures μ on $(\mathbb{X}, \mathfrak{X})$ with the property that $\mu(\mathcal{X}) < \infty$ for all bounded $\mathcal{X} \in \mathfrak{X}$ is a Polish space and its Borel σ-algebra is generated by the evaluation maps

$$\mathrm{eval}_{\mathcal{X}} \colon \mathcal{N}_{\mathbb{X}}^{\#} \to \mathbb{R} \colon \mu \mapsto \mu(\mathcal{X})$$

where $\mathcal{X} \in \mathfrak{X}$ [8, Proposition 9.1.IV]. The subspace $\mathcal{N}_{\mathbb{X}} = \mathrm{eval}_{\mathbb{X}}^{-1}(\mathbb{N})$ of measures of $\mathcal{N}_{\mathbb{X}}^{\#}$ of finite total mass is a measurable subset of $\mathcal{N}_{\mathbb{X}}^{\#}$ and thus, a standard Borel space when equipped with the corresponding trace σ-algebra [13, § 424G]. It is easy to see that there is a Borel isomorphism between our space $(\mathbb{D}, \mathfrak{D})$ and the space $(\mathcal{N}_{\mathbb{X}}, \mathcal{B}or(\mathcal{N}_{\mathbb{X}}))$. Since $\mathbb{D}_*$ is a measurable subset of $\mathbb{D}$, $(\mathbb{D}_*, \mathfrak{D}_*)$ is standard Borel as well. □

B BACKGROUND RESULTS FROM MEASURE THEORY

This section is intended to extend Section 2.1 by some well-known results. They can accordingly be found in the literature [25].

B.1 Measurability of Functions and Sets

The following statement says that collections of measurable functions yield a function that is measurable with respect to the product σ-algebra.

Fact B.1 ([25, Lemma 1.8, p. 5]). *If* $(\mathbb{X}, \mathfrak{X})$ *and* $(\mathcal{X}_i, \mathfrak{X}_i)$ *are measurable spaces (for* i *in some index set* I*) and* $f_i \colon \mathbb{X} \to \mathbb{X}_i$ *is measurable for all* $i \in I$*, then* $f \colon \mathbb{X} \to \prod_{i \in I} \mathcal{X}_i \colon x \mapsto (f_i(x))_{i \in I}$ *is* $\mathfrak{X}\text{-}\bigotimes_{i \in I} \mathfrak{X}_i$*-measurable.*

The next two results are concerned with the measurability of certain kinds of integration maps.

Fact B.2 ([25, Lemma 1.41(i), p. 21]). *Let* μ *be a stochastic kernel from* $\mathbb{X}$ *to* $\mathbb{Y}$ *and let* $f \colon \mathbb{X} \times \mathbb{Y} \to \mathbb{R}_{\geq 0}$ *be measurable. Then*

$$\mathbb{X} \to \mathbb{R}_{\geq 0} \colon x \mapsto \int f(x, \cdot)\ d\mu(x, \cdot)$$

is measurable.

Fact B.3 ([25, Lemma 1.26, p. 14]). *Let* $(\mathbb{X}, \mathfrak{X})$ *and* $(\mathbb{Y}, \mathfrak{Y})$ *be measurable spaces,* μ *a* σ*-finite measure on* $\mathbb{X}$ *and* $f \colon \mathbb{X} \times \mathbb{Y} \to \mathbb{R}_{\geq 0}$ *a measurable function.*

(i) The y-section $f(\cdot, y) \colon \mathbb{X} \to \mathbb{R}_{\geq 0}$ *of* f *is* $(\mathfrak{X}, \mathcal{B}or(\mathbb{R}_{\geq 0}))$*-measurable for all* $y \in \mathbb{Y}$*.*

(ii) The function $y \mapsto \int f(x, y)\ \mu(dx)$ *is* $(\mathfrak{Y}, \mathcal{B}or[0, 1])$*-measurable.*

If we have a measurable function between two standard Borel spaces, then the image of measurable sets needs not to be measurable in general, the standard example perhaps being projection functions (see [37, Proposition 4.1.1 and Theorem 4.1.5]. Given certain conditions however, measurable sets have measurable images under measurable functions:

Fact B.4 ([37, Theorem 4.5.4, p. 153]). *Let* $(\mathbb{X}, \mathfrak{X})$ *and* $(\mathbb{Y}, \mathfrak{Y})$ *be standard Borel,* $\mathcal{X} \in \mathfrak{X}$ *and let* $f \colon \mathcal{X} \to \mathbb{Y}$ *be an injective,* $(\mathfrak{X} \restriction \mathcal{X}, \mathfrak{Y})$*-measurable function. Then* $f(\mathcal{X}) \in \mathfrak{Y}$*.*

Finally, we come back to the multifunctions of Section 2.1.4 and explicitly state the theorem of Kuratowski and Ryll-Nardzewski on the existence of measurable selections:

Fact B.5 (Kuratowski and Ryll-Nardzewski [29], see [37, Theorem 5.2.1]). *Let* $(\mathbb{X}, \mathfrak{X})$ *be a measurable space and let* $(\mathbb{Y}, \mathcal{B}or(\mathbb{Y}))$ *be standard Borel. Then every closed-valued* $\mathfrak{X}$*-measurable multifunction* $M \colon \mathbb{X} \rightrightarrows \mathbb{Y}$ *has a* $(\mathfrak{X}, \mathcal{B}or(\mathbb{Y}))$*-measurable selection* $s \colon \mathbb{X} \to \mathbb{Y}$*.*

B.2 Identities for Integration

If μ is a measure and f a measurable function, then $f \cdot \mu := \nu$, defined by $\nu(\mathcal{X}) = \int_{\mathcal{X}} f\ d\mu$ is a measure. The following chain and substitution rules are the main tools to establish statements regarding the equality of transformed measures.

Fact B.6 (Chain Rule, cf. [25, Lemma 1.23, p. 12]). *Let* $(\mathbb{X}, \mathfrak{X}, \mu)$ *be a measure space and* $f \colon \mathbb{X} \to \mathbb{R}$ *and* $g \colon \mathbb{X} \to \mathbb{R}_{\geq 0}$ *be measurable function. Let* ν *be defined from* f *and* μ *like above (that is,* $\nu := f \cdot \mu$*). Then, if either of the following integrals exists (i. e. is finite), it holds that*

$$\int_{\mathbb{X}} f \cdot g\ d\mu = \int_{\mathbb{X}} g\ d\nu\,.$$

Fact B.7 (Substitution, cf. [25, Lemma 1.22, p.12]). *Let* $(\mathbb{X}, \mathfrak{X})$ *and* $(\mathbb{Y}, \mathfrak{Y})$ *be measurable spaces and* μ *a measure on* $(\mathbb{X}, \mathfrak{X})$*. Let* $f \colon \mathbb{X} \to \mathbb{Y}$ *and* $g \colon \mathbb{Y} \to \mathbb{R}$ *be measurable. Then, if either of the following integrals exists (i. e. is finite), it holds that*

$$\int_{\mathbb{X}} g \circ f\ d\mu = \int_{\mathbb{Y}} g\ d(\mu \circ f^{-1})$$

where $\mu \circ f^{-1}$ *is the push-forward measure of* μ *along* f *on* $(\mathbb{Y}, \mathfrak{Y})$*.*

Another such main tool is Fubini's Theorem, stating, in a nutshell that integration in a product space can be carried out in an arbitrary order.

Fact B.8 (Fubini's Theorem, cf. [25, Theorem 1.27]). *Let* $(\mathbb{X}, \mathfrak{X})$ *and* $(\mathbb{Y}, \mathfrak{Y})$ *be measurable spaces and* μ *be a* σ*-finite measure on* $(\mathbb{X}, \mathfrak{X})$*. Then for all measurable* $f \colon \mathbb{X} \times \mathbb{Y} \to \mathbb{R}_{\geq 0}$*, it holds that*

$$\int_{\mathbb{X} \times \mathbb{Y}} f\ d(\mu \otimes \nu) = \int_{\mathbb{X}} \left(\int_{\mathbb{Y}} f\ d\nu \right) d\mu = \int_{\mathbb{Y}} \left(\int_{\mathbb{X}} f\ d\mu \right) d\nu\,.$$

B.3 Existence of Markov Processes

Fact B.9 (Existence of Markov Processes, Kolmogorov, cf. [25, Theorem 8.4]). *Let* $(\mathbb{X}, \mathfrak{X})$ *be a standard Borel space,* μ_0 *a probability measure on* $(\mathbb{X}, \mathfrak{X})$ *and* $(\mu_i)_{i \geq 1}$ *a family of stochastic kernels* $\mu_i \colon \mathbb{X} \times \mathfrak{X} \to [0, 1]$ *for* $i \geq 1$*. Then there exists a Markov process* ξ *(with time scale* $\mathbb{N}$ *and paths in* $\prod_{i=0}^{\infty} \mathbb{X}$*) with initial distribution* μ_0 *and transition kernels* μ_i*.*