Gamma Probabilistic Databases: Learning from Exchangeable Query-Answers

Niccolò Meneghetti
niccolom@umich.edu
University of Michigan-Dearborn
Dearborn, Michigan, USA

Ouael Ben Amara
benamara@umich.edu
University of Michigan-Dearborn
Dearborn, Michigan, USA

ABSTRACT

In this paper we propose a novel knowledge compilation technique that compiles *Bayesian inference procedures*, starting from probabilistic programs expressed in terms of probabilistic query-answers. To do so, we extend the framework of Dirichlet Probabilistic Databases with the ability to process *exchangeable* observations of query-answers. We show that the resulting framework can encode non-trivial models, like Latent Dirichlet Allocation and the Ising model, and generate high-performance Gibbs samplers for both models.

1 INTRODUCTION

Implementing inference algorithms for Bayesian statistics is difficult and expensive. It requires a deep knowledge of complex topics in statistics, like Markov Chain Monte Carlo methods [21] or variational inference [5], but also the ability to write code that is efficient, scalable, and easy to verify and maintain in the long term. As a result, most data scientists do not implement their own inference algorithms from scratch, but rather rely on special-purpose environments, such as Python[30], Julia [3], R [32] and many other frameworks that are explicitly targeted at statistical modeling and inference.

When it comes to Bayesian statistics, Probabilistic Programming [67] is one of the most popular paradigms to ease the work of data scientists. A probabilistic program consists of two components: (i) the definition of a stochastic generative model and (ii) a set of observations for such model. Together, the two components define a posterior distribution over the latent variables of the generative process, obtained by conditioning the process w.r.t. the observations. Given a generative model and a set of observations, a Probabilistic Programming compiler can build an inference algorithm for the posterior distribution, without requiring any additional implementation effort from the end-user. This paradigm has been very successful, giving rise to languages like Stan [10], Edward [65], PyMC3 [57], Pyro [4] and many others [2, 25, 43, 47, 61, 64].

Most Probabilistic Programming frameworks are developed to operate as independent and autonomous tools, with limited capabilities to integrate with existing database systems. They often adopt non-relational data models to encode the training data, and use imperative languages to define the stochastic generative processes. This is in stark contrast with standard database systems, where the data is strongly relational and the processing is defined in terms of declarative query-languages. As a result, the two systems are often deployed in a loosely coupled configuration, where the database is used exclusively for data storage, indexing and retrieval, while the analytical tool is forced to operate on replicas of the data, exported from the database. This configuration is both inefficient and cumbersome to maintain [7]. A better integration would be very desirable: legacy database systems often store sensitive data that was expensive to collect in the first place; it would be beneficial to move the computation close to where such data resides. Beyond efficiency considerations, database systems offer a number of features that are hard to replicate by other systems. These include the support for transactions, the use of standardized policies for access control, resource management, data replication, data partitioning, indexing, normalization and compression, the clean separation between modeling and computation, the extensive use of cost-based optimization. Furthermore, database systems can exploit knowledge about the structural properties of both the data [52] and the queries [38] to speed-up processing. The challenges and opportunities posed by the integration of machine learning processing into standard database systems are discussed in great detail in [7, 33, 51].

With the goal of narrowing the gap between database systems and statistical analysis tools, in this paper we introduce a novel, database-friendly Probabilistic Programming framework. Our approach is database-friendly in the sense that we encode the training data in relational form, and express the stochastic processes in terms of probabilistic query-answers. Probabilistic queries are just regular relational queries that are run against a probabilistic database [63], a database that contains both deterministic and probabilistic data [16, 19]. The answer to such queries will vary depending on the values taken by the database latent random variables. Thus, a probabilistic query defines a stochastic generative process for all its plausible answers and, when paired with a set of observed query-answers, it determines a well-formed probabilistic program.

In this paper we make two main contributions:

(1) We introduce a novel data model to express probabilistic programs in terms of query-answers.
(2) We propose a knowledge compilation technique to compile probabilistic programs into inference methods.

To achieve the first goal, we extend the data model adopted by Dirichlet Probabilistic Databases [46] with the ability to process *exchangeable observations* [15] of query-answers. Two query-answers are said to be exchangeable if they are stochastically independent when the state of the underlying probabilistic database is known, but dependent when the state is unknown. Thus, exchangeability is a relaxation of the assumption of independent and identically distributed (i.i.d.) query-answers made in [46]. This is a radical extension of [46], since the new data model can account for correlations across multiple observed query-answers, but also variate the number of the internal latent variables in a dynamic fashion, as a function of the observations [48].

To achieve the second goal, we devise a novel knowledge compilation method, inspired by [20], that translates a collection of exchangeable query-answers into a Gibbs sampler [23] for the corresponding posterior distribution. This paper does

not describe the implementation of a new database system. Our contribution consists of the formulation of a well-founded data model for exchangeable query-answers and the development of knowledge compilation technique for supporting inference. We show that the two components are powerful enough to represent non-trivial models, like Latent Dirichlet Allocation [6] and the Ising model [41], and to compile high-performance Gibbs samplers for both models.

2 PRELIMINARIES

A probabilistic database [63] is a database that stores uncertain data. For ease of presentation, and without lack of generality, we assume that uncertainty only affects certain attributes in the database schema. Figure 1 depicts a simple probabilistic database that stores information about a set of employees (Ada and Bob); the database contains two uncertain attributes, one for defining the roles of the employees (either tech-lead, developer, or QA engineer) and another for the employees' level of experience (either junior or senior). Each probabilistic tuple can be seen as a random choice, that decides the value assigned to an uncertain attribute. The database prescribes a probability distribution for all these choices: the parameters in $\Theta = \{\theta_{i,j}\}_{i,j}$ define the likelihood of each assignment. For example, the likelihood of the assignment *Role[Ada] = Lead* is 1/3 according to the probabilistic database in Figure 1. A *possible world* is a collection of assignments that covers all the random tuples. The database in Figure 1 consists of four probabilistic tuples, for a total of 36 possible worlds. It is easy to realize that that each possible world represents a deterministic database instance. Furthermore, the database defines a probability distribution over the set of all possible worlds.

In this paper we use *query-answers* to identify subsets of possible worlds. A query-answer simply represents the belief that a certain relational query should return a certain answer, i.e. a certain (deterministic) relation instance. For example, if we want to identify all the possible worlds where only senior employees can take the role of tech-leads, we can do so by means of the following query-answer (q_1):

$$\sigma_{\text{role=Lead}\wedge\text{exp}\neq\text{Senior}}(R \bowtie S) \subseteq \emptyset \tag{1}$$

Similarly, we can express the belief that Ada is either a developer or a QA engineer using the following query-answer (q_2):

$$\sigma_{\text{emp=Ada}\wedge\text{role=Lead}}(R) \subseteq \emptyset \tag{2}$$

The first query-answer identifies a subset of 25 possible worlds, while the second identifies another subset of 24 possible worlds. Using *lineage* [26] expressions, we can represent each query-answer with a Boolean formula. For example, all the possible worlds identified by q_1 must satisfy the following Boolean expression:

$$((\text{Role[Ada]} \neq \text{Lead}) \vee (\text{Exp[Ada]} = \text{Senior})) \ \wedge \ldots$$
$$\ldots \wedge \ ((\text{Role[Bob]} \neq \text{Lead}) \vee (\text{Exp[Bob]} = \text{Senior}))$$

Similarly, query-answer q_2 can be expressed as follows:

$$(\text{Role[Ada]} \neq \text{Lead})$$

The parameters in Θ determine the probability of observing a certain query-answer. For example, the likelihood of observing q_1 is given by

$$P[q_1|\Theta] = [1 - (\theta_{1,1} \cdot (1 - \theta_{3,1}))] \cdot [1 - (\theta_{2,1} \cdot (1 - \theta_{4,1}))]$$

Similarly, the likelihood of query-answer q_2 is given by

$$P[q_2|\Theta] = (1 - \theta_{1,1})$$

Let's imagine that two independent observers sample two possible worlds from our probabilistic database, and that the possible world sampled by the first observer belongs to q_1, while the possible world sampled by the second observer belongs to q_2. Under these assumptions, we say that q_1 and q_2 are two *exchangeable* [15] query-answers. When the parameters in Θ are known quantities, the two observations represent two stochastically independent events. In other words $P[q_2|\Theta, q_1] = P[q_2|\Theta] = 2/3$. The same is *not* true when Θ is *not* fully known. For example, let's now assume that $\boldsymbol{\theta}_1 = (\theta_{1,1}, \theta_{1,2}, \theta_{1,3})$ is a latent random vector, uniformly distributed over the three-dimensional probabilistic simplex. Under these assumptions, the probability of q_2 conditioned on q_1 can be computed as follows:

$$P[q_2|\Theta \setminus \{\boldsymbol{\theta}_1\}, q_1] = \int_{\text{Dom}(\boldsymbol{\theta}_1)} P[q_2|\boldsymbol{\theta}_1] \cdot P[\boldsymbol{\theta}_1|\Theta \setminus \{\boldsymbol{\theta}_1\}, q_1] \, d\boldsymbol{\theta}_1 \approx 0.74$$

Therefore $P[q_2|\Theta\setminus\{\boldsymbol{\theta}_1\}, q_1] \neq P[q_2|\Theta\setminus\{\boldsymbol{\theta}_1\}]$ and we can conclude that q_1 and q_2 are not independent.

The main goal of this paper is to show how to learn the parameters of a probabilistic database from observations, using a collection of exchangeable query-answers as training data.

The remainder of this paper is organized as follows: in Section 2.1 we discuss some required background knowledge in predicate logic; we introduce the concept of *dynamic Boolean expression* (Section 2.2), the base building block for supporting dynamic variable allocation [48] in our data model. We show how to use dynamic Boolean expressions to describe the state of a collection of exchangeable latent variables (Section 2.4) and later we apply these new concepts to extend the probabilistic database model from [46], formalizing the definition of "Gamma Probabilistic Database" (Section 3). We conclude our discussion with experimental results (Section 4).

Roles (R)

emp	role	Θ
Ada	Lead	$\theta_{1,1} = 1/3$
Ada	Dev	$\theta_{1,2} = 1/3$
Ada	QA	$\theta_{1,3} = 1/3$
Bob	Lead	$\theta_{2,1} = 1/7$
Bob	Dev	$\theta_{2,2} = 3/7$
Bob	QA	$\theta_{2,3} = 4/7$

Seniority (S)

emp	exp	Θ
Ada	Senior	$\theta_{3,1} = 1/10$
Ada	Junior	$\theta_{3,2} = 9/10$
Bob	Senior	$\theta_{4,1} = 1/2$
Bob	Junior	$\theta_{4,2} = 1/2$

Figure 1: A simple probabilistic database.

2.1 Boolean Expressions

Boolean variables take values in the set $\mathcal{B} = \{\top, \bot\}$. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set of n Boolean variables; we denote by $\text{Asst}(X)$ the set of all possible assignments to all the variables in X. A *Boolean function* [12] over X is a function from $\text{Asst}(X)$ into $\mathcal{B}$. Boolean functions can be represented as *Boolean expressions*, sentences generated by the following grammar:

$$\phi \ ::= \ x_i = \top \ \mid \ x_i = \bot \ \mid \ \neg\phi_1 \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \phi_1 \vee \phi_2 \tag{3}$$

A Boolean expression may consist of any combination of literals (value-assignments to Boolean variables), logical disjunctions of expressions ($\phi_1 \vee \phi_2$), logical conjunctions ($\phi_1 \wedge \phi_2$), or logical negations ($\neg\phi_1$). For the sake of brevity we will often omit the value-assignment symbol (=), writing x_i and $\overline{x}_i$ in place of $x_i = \top$

and $x_i = \bot$, respectively. Similarly, we will often omit the logical conjunction symbol ($\wedge$), writing $\phi_1\phi_2$ in place of $\phi_1 \wedge \phi_2$. A *term* is an expression that consists of a conjunction of literals, a *clause* is an expression that consists of a disjunction of literals. We will often represent the elements in $\textsc{Asst}(X)$ as term-expressions over X.

A Boolean expression is in *negation normal form* (or NNF) if it is negation-free, except for its literals [14]. It is in disjunctive normal form (DNF) if it consists of a disjunction of terms; it is in conjunctive normal form (CNF) if it consists of a conjunction of clauses. A Boolean expression is *read-once* (RO) if each of its variables appears in at most one literal [24]. A Boolean function is read-once if it admits a RO representation. Verifying if such representation exists takes polynomial time in the size of the DNF representation of the function [24]. Converting an arbitrary Boolean expression into NNF is always possible and takes linear time in the size of the expression. Furthermore, RO expressions remain such when converted to NNF.

We denote by $\textsc{Var}(\phi)$ the set of variables that appear in expression ϕ as literals. Let X be a set of variables that contains $\textsc{Var}(\phi)$, we denote by $\textsc{Sat}(\phi, X)$ the subset of $\textsc{Asst}(X)$ where ϕ evaluates to $\top$, i.e. all the assignments that *satisfy* expression ϕ. Notice that X may contain some variables that do not appear in expression ϕ. The same Boolean function may be represented by many different Boolean expressions. When two expressions ϕ_1 and ϕ_2 represent the same Boolean function, we say that they are *logically equivalent* and write $\phi_1 = \phi_2$. We say that two expressions are *independent* when they do not share any variable. We say that they are *mutually exclusive* if every assignment that satisfies one expression never satisfies the other. We say that expression ϕ_1 *entails* expression ϕ_2 (and write $\phi_1 \models \phi_2$) when the expression $\neg\phi_1 \vee \phi_2$ always evaluates to $\top$ (that is, when every assignment that satisfies ϕ_1 also satisfies ϕ_2). The concepts of satisfiability, logical equivalence, entailment, mutual exclusion and independence are similarly defined for Boolean functions.

If ϕ is a Boolean expression and x_i is one of its variables, we denote by $\phi||x_i$ the expression obtained by replacing all the instances of variable x_i in ϕ with $\top$, and simplifying the resulting expression by applying the following logical equivalences: (i) $(\top \wedge \phi) = \phi$, (ii) $(\bot \wedge \phi) = \bot$, (iii) $(\top \vee \phi) = \top$, (iv) $(\bot \vee \phi) = \phi$, (v) $\neg(\top) = \bot$, (vi) $\neg(\bot) = \top$. Similarly, we denote by $\phi||\overline{x}_i$ the expression obtained by replacing all the instances of x_i with $\bot$. If τ is a term-expression over some variables of ϕ, then $\phi||\tau$ denotes the formula obtained by sequentially replacing in ϕ all the variables that appear in τ. We say that variable x_i in *inessential* in expression ϕ if $\textsc{Sat}(\phi||x_i, X) = \textsc{Sat}(\phi||\overline{x}_i, X)$. If x_i in inessential, then ϕ can be rewritten without using x_i. If ϕ is a Boolean expression, we denote by $[\phi]$ its *indicator function*, a function from $\textsc{Asst}(\textsc{Var}(\phi))$ into $\{0, 1\}$ that evaluates to 1 when ϕ is satisfied and 0 otherwise.

Let ϕ be an expression where some variable x_i appears more than once. A *Boole-Shannon expansion* [60] allows us to rewrite ϕ as the disjunction of two mutually exclusive expressions: $\phi = (x_i \wedge \phi||x_i) \vee (\overline{x}_i \wedge \phi||\overline{x}_i)$. Notice that after the expansion variable x_i appears only once in each term of the disjunction.

In the remainder of this paper we will extend the grammar of Boolean expressions to allow for the use of *categorical variables*. A categorical variable x_i is a variable that takes values in some finite, discrete domain $\textsc{Dom}(x_i) = \{v_1, \ldots, v_c\}$ with cardinality c possibly larger than 2. Categorical literals take the form $(x_i \in V)$, where V is a non-empty subset of $\textsc{Dom}(x_i)$. When V contains a single element, say v_j, we will simply write $x_i = v_j$. It is easy to verify that categorical literals respect the following logical equivalences: (i) $(x_i \in V_1) \wedge (x_i \in V_2) = (x_i \in V_1 \cap V_2)$, (ii) $(x_i \in V_1) \vee (x_i \in V_2) = (x_i \in V_1 \cup V_2)$, (iii) $\neg(x_i \in V) = (x_i \in (\textsc{Dom}(x_i) - V))$, (iv) $(x_i \in \textsc{Dom}(x_i)) = \top$, (v) $(x_i \in \emptyset) = \bot$. If ϕ is a Boolean expression where each variable, no matter if Boolean or categorical, appears at most once, then we say that ϕ is read-once. Furthermore, we denote by $\phi||x_i \in V^*$ the expression obtained by replacing all the literals in the form $(x_i \in V)$ with $\top$, whenever $V \cap V^* \neq \emptyset$, and with $\bot$ otherwise. It is straightforward to generalize Boole-Shannon expansions w.r.t. categorical variables: $\phi = \bigvee_{v_j \in \textsc{Dom}(x_i)}((x_i = v_j) \wedge (\phi||x_i = v_j))$. We say that a categorical variable x_i is *inessential* in expression ϕ whenever $\textsc{Sat}(\phi||x_i = v, X) = \textsc{Sat}(\phi||x_i = v', X)$, for every v and v' in $\textsc{Dom}(x_i)$. For economy of notation, in the remainder of this paper we will treat Boolean variables as categorical variables having domain cardinality c equal to 2.

D-trees [20] are NNF expressions where conjunctions are only allowed between subexpressions that are independent, and disjunctions are only allowed between subexpressions that are either independent or mutually exclusive. We represent d-trees as sentences generated by the following grammar:

$$\psi \quad ::= \quad x_i \in V \;\mid\; \psi_1 \odot \psi_2 \;\mid\; \psi_1 \otimes \psi_2 \;\mid\; \oplus^{x_i}(\psi_1, \ldots, \psi_k) \quad (4)$$

If $\{\psi_1, \ldots, \psi_k\}$ is a collection of k d-trees that represent the Boolean expressions $\{\phi_1, \ldots, \phi_k\}$, then the d-tree $\oplus^{x_i}(\psi_1, ..., \psi_k)$ represents the disjunction $\bigvee_{j=1}^{k} \phi_j$, under the assumption that $\phi_j \models (x_i = v_j)$ holds true for every $j \in \{1, \ldots, k\}$. Any Boolean expression can be represented as a d-tree. Algorithm 1, proposed in [20], shows a way to translate arbitrary CNF expressions into well-formed d-trees.

The symbol $\odot$ (respectively, $\otimes$) represents a conjunction (disjunction) between independent expressions. The symbol $\oplus$ represents a disjunction between mutually exclusive expressions. If d-trees ψ_1 and ψ_2 represent the Boolean expressions ϕ_1 and ϕ_2, respectively, then the d-tree $(\psi_1 \odot \psi_2)$ represents the expression $(\phi_1 \wedge \phi_2)$, while the the d-tree $(\psi_1 \otimes \psi_2)$ represents the expression $(\phi_1 \vee \phi_2)$, under the assumption that ϕ_1 and ϕ_2 are independent. Intuitively, Algorithm 1 converts the input Boolean expression ϕ into a collection of mutually-exclusive, read-once expressions, by repeatedly applying the Boole-Shannon expansion to the variables that appear more than once (lines 3–6). All the disjunctions and conjunctions in the resulting read-once expressions are then translated into the $\otimes$ and $\odot$ operators, since they always combine subexpressions that are pairwise independent.

Algorithm 1: CompileDTree: Compilation of Boolean expressions into d-tree expressions. Adapted from [20].

Input: A Boolean expression ϕ, in CNF.
Output: A d-tree expression ψ that represents ϕ.

```
1  CompileDTree(φ):
2      Remove redundant clauses from φ.
3      if variable x appears more than once in φ then
4          for v_j ∈ Dom(x) do
5              ψ_j ← CompileDTree(φ||x = v_j)
6          return ⊕^x(((x = v_1) ⊙ ψ_1), ..., ((x = v_k) ⊙ ψ_k))
7      else if φ = φ_1 ∨ φ_2 and φ_1 and φ_2 are independent then
8          return CompileDTree(φ_1) ⊗ CompileDTree(φ_2)
9      else if φ = φ_1 ∧ φ_2 and φ_1 and φ_2 are independent then
10         return CompileDTree(φ_1) ⊙ CompileDTree(φ_2)
11     else if φ is a literal or a constant then
12         return φ
```

Notice that the same Boolean expression may be represented by several different d-trees, depending on the order in which the Boole-Shannon expansions are performed. For example, the DNF expression $x_1x_2x_3 \vee \overline{x}_1\overline{x}_2x_4 \vee x_1x_5$ may be represented as $\oplus^{x_1}(((x_2 \odot x_3) \otimes x_5), (\overline{x}_2 \odot x_4))$ or as $\oplus^{x_2}((x_1 \odot (x_3 \otimes x_5)), (\overline{x}_1 \odot x_4))$.

Definition 1 (Almost Read-Once expressions). *A d-tree ψ is* almost read-once (ARO) *if the operator $\otimes$ is only applied to read-once subexpressions.*

Notice that Algorithm 1, by design, always generates ARO expressions. Every Boolean expression can be compiled into an almost read-once d-tree, but the size of the d-tree can grow exponentially large w.r.t. the size of the expression.

2.2 Dynamic Boolean Expressions

In this paper we introduce the concept of *dynamic Boolean expression*. A dynamic expression is a regular Boolean expression defined over the union of two disjoint sets of variables, the *regular* variables $X = \{x_1, \ldots, x_n\}$ and the *volatile* variables $Y = \{y_1, \ldots, y_m\}$. Each volatile variable y_j is associated with an *activation condition* $\mathrm{AC}(y_i)$, a Boolean expression over the variables in $(X \cup Y) - \{y_i\}$. We say that a volatile variable y_j is *active* whenever its activation condition is satisfied. Regular variables in X are considered to be always active. A dynamic expression ϕ must satisfy the following two properties: (i) for every volatile variable y_i and every assignment $\tau \in \mathrm{Sat}(\neg\mathrm{AC}(y_i), \mathrm{Var}(\mathrm{AC}(y_i)))$ that leaves it inactive, y_i must be inessential w.r.t. expression $\phi||\tau$, (ii) if any volatile variable y_i is essential in the activation expression of some other volatile variable y_j, then $\mathrm{AC}(y_j) \models \mathrm{AC}(y_i)$ must hold true. Property (ii) induces a natural partial order $\prec_a$ over the volatile variables, that describes the evaluation dependencies across the activation conditions: if $R(y_i, y_j) \subset Y^2$ is the relation that associates each volatile variable y_j with all the other volatile variables y_i that are essential in its activation expression, we define $\prec_a$ as the transitive closure of R. It is easy to verify that $\prec_a$ is transitive, asymmetric and irreflexive, and that $y_i \prec_a y_j$ entails $\mathrm{AC}(y_j) \models \mathrm{AC}(y_i)$.

From now on we use the tuple (ϕ, X, Y) to define a dynamic Boolean expression ϕ with regular variables X and volatile variables Y. We denote by $\mathrm{Dsat}(\phi, X, Y)$ the set of assignments that satisfy ϕ where all variables are active. More precisely, we define $\mathrm{Dsat}(\phi, X, Y)$ as the set of term-expressions $\{\tau_1, \ldots, \tau_m\}$ that satisfy the following properties:

(1) $\forall \tau \in \mathrm{Dsat}(\phi, X, Y) \quad X \subseteq \mathrm{Var}(\tau) \subseteq X \cup Y$
(2) $\forall \tau \in \mathrm{Dsat}(\phi, X, Y) \quad \tau \models \phi$
(3) $\forall \tau' \in \mathrm{Sat}(\phi) \quad \exists \tau \in \mathrm{Dsat}(\phi, X, Y) \quad \tau' \models \tau$
(4) $\forall \tau \in \mathrm{Dsat}(\phi, X, Y) \quad$ if $y \in \mathrm{Var}(\tau) \cap Y$ then $\tau \models \mathrm{AC}(y)$
(5) $\forall \tau \in \mathrm{Dsat}(\phi, X, Y) \quad$ if $y \in Y - \mathrm{Var}(\tau)$ then $\tau \models \neg\mathrm{AC}(y)$

For example, if $\phi = (x_1 \vee x_2) \wedge (\overline{x}_1 \vee y_1)$ and $\mathrm{AC}(y_1) = x_1$, then ϕ is a valid dynamic expression and $\mathrm{Dsat}(\phi, \{x_1, x_2\}, \{y_1\})$ is equal to $\{x_1x_2y_1,\ \overline{x}_1x_2,\ x_1\overline{x}_2y_1\}$. Notice that this set provides a compact representation of $\mathrm{Sat}(\phi, \{x_1, x_2, y_1\})$, where variable y_1 is ignored when inactive and therefore inessential.

Proposition 1. *All the terms in $\mathrm{Dsat}(\phi, X, Y)$ are mutually exclusive.*

$$\forall \tau, \tau' \in \mathrm{Dsat}(\phi, X, Y)\ \ \tau \neq \tau' \Rightarrow \tau \models \neg\tau' \tag{5}$$

Proof. Let τ and τ' be two distinct terms in $\mathrm{Dsat}(\phi, X, Y)$. Let's assume, by contradiction, that $\tau \wedge \tau'$ is satisfiable (i.e. the two terms are not mutually exclusive). It follows that $\mathrm{Var}(\tau)$ must be different from $\mathrm{Var}(\tau')$. By property (1) we can conclude that there is at least one volatile variable that is active in one term and inactive in the other. Let y be that variable. By properties (4) and (5) we can conclude that one of the terms entails $\mathrm{AC}(y)$, while the other entails $\neg\mathrm{AC}(y)$. This contradicts our initial assumption that τ and τ' are not mutually exclusive. □

Proposition 2. *The disjunction of the terms in $\mathrm{Dsat}(\phi, X, Y)$ is logically equivalent to the disjunction of the terms in $\mathrm{Sat}(\phi, X \cup Y)$.*

$$\vee_{\tau \in \mathrm{Dsat}(\phi, X, Y)}\ \tau = \vee_{\tau' \in \mathrm{Sat}(\phi, X \cup Y)} \tau' \tag{6}$$

Proof. Let's denote by ϕ_1 the expression $(\vee_{\tau \in \mathrm{Dsat}(\phi, X, Y)} \tau)$, and by ϕ_2 the expression $\vee_{\tau' \in \mathrm{Sat}(\phi, X \cup Y)} \tau'$. Since ϕ_2 is logically equivalent to ϕ, by property (2) we can conclude that $\phi_1 \models \phi_2$. Let's now assume, by contradiction, that ϕ_2 does not entail ϕ_1. This means that there is at least one term τ' in $\mathrm{Sat}(\phi, X \cup Y)$ that does not satisfy ϕ_1, nor any of the term expressions τ in $\mathrm{Dsat}(\phi, X, Y)$. This contradicts property (3), that states that there is at least one term τ in $\mathrm{Dsat}(\phi, X, Y)$ such that $\tau' \models \tau$. We can conclude that $\phi_2 \models \phi_1$ and that the two expressions are logically equivalent. □

Proposition 3. *Let (ϕ_1, X_1, Y_1) and (ϕ_2, X_2, Y_2) be two dynamic Boolean expressions that do not share any variable (i.e. $(X_1 \cup Y_1) \cap (X_2 \cup Y_2) = \emptyset$). Let $X = X_1 \cup X_2$ and $Y = Y_1 \cup Y_2$. The conjunction of ϕ_1 and ϕ_2 is a well-defined dynamic Boolean expression $(\phi_1 \wedge \phi_2, X, Y)$.*

Proof. In order to prove the thesis, we need to identify an activation condition for each volatile variable in $Y = Y_1 \cup Y_2$. For the variables in Y_1 we keep the original activation conditions from expression (ϕ_1, X_1, Y_1); similarly, for Y_2 we use the original activation conditions from expression (ϕ_2, X_2, Y_2). In the resulting dynamic expression $(\phi_1 \wedge \phi_2, X, Y)$ the set $\mathrm{Dsat}(\phi_1 \wedge \phi_2, X, Y)$ is equivalent to the set $\{\tau_1 \wedge \tau_2 | (\tau_1, \tau_2) \in \mathrm{Dsat}(\phi_1, X_1, Y_1) \times \mathrm{Dsat}(\phi_2, X_2, Y_2)\}$. The thesis follows immediately. □

Proposition 4. *Let (ϕ_1, X, Y_1) and (ϕ_2, X, Y_2) be two mutually exclusive dynamic Boolean expressions that do not share any volatile variable ($Y_1 \cap Y_2 = \emptyset$).*

If every term in $\mathrm{Dsat}(\phi_1, X, Y_1)$ leaves all the variables in Y_2 inactive, and every term in $\mathrm{Dsat}(\phi_2, X, Y_2)$ leaves all the variables in Y_1 inactive, then the disjunction $(\phi_1 \vee \phi_2, X, Y_1 \cup Y_2)$ is a well-defined dynamic Boolean expression.

Proof. If we assign to the volatile variables in $Y = Y_1 \cup Y_2$ the same activation conditions as in expressions (ϕ_1, X, Y_1) and (ϕ_2, X, Y_2), then $\mathrm{Dsat}(\phi_1 \vee \phi_2, X, Y_1 \cup Y_2)$ consists of the union of $\mathrm{Dsat}(\phi_1, X, Y_1)$ and $\mathrm{Dsat}(\phi_2, X, Y_2)$. □

In order to represent dynamic Boolean expressions as d-trees, we extend the grammar from Equation 4 with an additional operator $\oplus^{\mathrm{AC}(y)}$: if ψ_1 and ψ_2 are two d-trees representing the expressions ϕ_1 and ϕ_2, respectively, then the d-tree $\oplus^{\mathrm{AC}(y)}(\psi_1, \psi_2)$ represents the disjunction $\phi_1 \vee \phi_2$, under the assumption that $\phi_1 \models \neg\mathrm{AC}(y)$ and $\phi_2 \models \mathrm{AC}(y)$ hold true, and that variable y is inessential in ϕ_1. We define a *dynamic d-tree* as a collection of regular d-trees joint by the $\oplus^{\mathrm{AC}(y)}$ operator.

Proposition 5. *Every dynamic Boolean expression ϕ with regular variables X, volatile variables Y, and activation conditions $AC(\cdot)$ can be represented as a dynamic d-tree that satisfies the* ARO *property.*

Algorithm 2: CompileDynDTree: Compilation of dynamic Boolean expressions into dynamic d-trees.

Input: A dynamic Boolean expression (ϕ, X, Y).
Output: A dynamic d-tree expression (ψ, X, Y).

1 *CompileDynDTree*(ϕ, X, Y):
2 **if** $Y \neq \emptyset$ **then**
3 Let y be one of the maximal elements of Y w.r.t. $\prec_a$
4 $\psi_1 \leftarrow$ CompileDynDTree$((\neg \text{AC}(y) \wedge \phi), X, Y - \{y\})$
5 $\psi_2 \leftarrow$ CompileDynDTree$((\text{AC}(y) \wedge \phi), X \cup \{y\}, Y - \{y\})$
6 **return** $(\oplus^{\text{AC}(y)}(\psi_1, \psi_2), X, Y)$
7 **else**
8 **return** (CompileDTree$(\phi), X, \emptyset)$

Proof. Algorithm 2 (CompileDynDTree) transforms an arbitrary dynamic Boolean expression into a well-formed dynamic d-tree. At lines 2–6 the algorithm recursively partitions $\text{Asst}(X \cup Y)$ into a collection of disjoint subsets, so that all the assignments in each subset all share the very same active variables. For each of these subsets, the algorithm compiles a regular d-tree expression over the appropriate active variables (line 8). Since the algorithm invokes the CompileDTree procedure, the resulting d-tree expression is guaranteed to be almost read-once. Notice that the disjunction $\oplus^{\text{AC}(y)}(\psi_1, \psi_2)$ at line 6 does not represent a regular Boole-Shannon expansion, since variable y is made inessential (i.e. eliminated) only in ψ_1, and not in ψ_2.

□

2.3 Boolean Expressions over Statistically Independent Random Variables

If x_i is a categorical variable with domain $\text{Dom}(x_i) = \{v_1, \ldots, v_c\}$ and $\boldsymbol{\theta}_i = (\theta_{i,1}, \ldots, \theta_{i,c})$ is a c-vector of non-negative real numbers that sum up to 1, we denote by $P[x_i|\boldsymbol{\theta}_i]$ the probability mass function of a categorical distribution, that assigns value v_j to variable x_i with probability $\theta_{i,j}$

$$P[x_i|\boldsymbol{\theta}_i] = \prod_{j=1}^{c} \theta_{i,j}^{[x_i = v_j]} \tag{7}$$

We can think at x_i as a categorically distributed random variable with parameters $\boldsymbol{\theta}_i$. Notice that when $c = 2$, Equation 7 represents the probability mass function of a Bernoulli distribution over the domain of a Boolean variable. Let $X = \{x_i\}_i$ be a collection of Bernoulli and categorical random variables parametrized by $\Theta = \{\boldsymbol{\theta}_i\}_i$. It is easy to derive from Equation 7 a probability distribution over the elements of $\text{Asst}(X)$. Let τ be an arbitrary term-expression in $\text{Asst}(X)$, its probability can be computed as follows

$$P[\tau|\Theta] = \prod_{i=1}^{n} \prod_{j=1}^{c} \theta_{i,j}^{[\tau \models (x_i = v_j)]} \tag{8}$$

Notice that Equation 8 states that all the random variables in X are *statistically independent*, i.e. for every pair of distinct variables x_1 and x_2, $P[x_1, x_2|\Theta] = P[x_1|\Theta] \cdot P[x_2|\Theta]$. If ϕ is a Boolean expression over X, we define $P[\phi|\Theta]$ as the probability of sampling from $\text{Asst}(X)$ an assignment that satisfies ϕ.

$$P[\phi|\Theta] = \sum_{\tau \in \text{Sat}(\phi, X)} P[\tau|\Theta] \tag{9}$$

Computing $P[\phi|\Theta]$ is known to be $\#P$-hard in the size of ϕ [66]. If ψ is a d-tree expression representing ϕ, computing $P[\phi|\Theta]$ takes polynomial time in the size of ψ [20]. Algorithm 3, adapted from

Algorithm 3: ProbDTree: Computation of the probability of a d-tree expression being satisfied. Adapted from [20].

Input: A d-tree expression ψ, a probability distribution $P[\cdot|\Theta]$.
Output: The probability $P[\psi|\Theta]$.

1 *ProbDTree*(ψ, Θ):
2 **if** $\psi = (x_i = v_j)$ **then**
3 **return** $\theta_{i,j}$
4 **else if** $\psi = \psi_1 \odot \psi_2$ **then**
5 **return** ProbDTree$(\psi_1, \Theta) \cdot$ ProbDTree(ψ_2, Θ)
6 **else if** $\psi = \psi_1 \otimes \psi_2$ **then**
7 **return** $1 - [(1 - \text{ProbDTree}(\psi_1, \Theta)) \cdot (1 - \text{ProbDTree}(\psi_2, \Theta))]$
8 **else if** $\psi = \psi_1 \oplus \psi_2$ **then**
9 **return** ProbDTree$(\psi_1, \Theta) +$ ProbDTree(ψ_2, Θ)

[20], shows how to compute $P[\psi|\Theta]$. Notice that the algorithm works seamlessly with both regular and dynamic d-tree expressions. If ϕ_1 and ϕ_2 are two Boolean expressions over X, we denote by $P[\phi_1|\phi_2, \Theta]$ the probability of sampling from $\text{Sat}(\phi_2, X)$ an assignment that also satisfies ϕ_1. Such probability can be computed as follows

$$P[\phi_1|\phi_2, \Theta] = \frac{P[\phi_1 \wedge \phi_2|\Theta]}{P[\phi_2|\Theta]} \tag{10}$$

We can use Equation 10 to define a proper probability distribution over $\text{Sat}(\phi, X)$, the subset of $\text{Asst}(X)$ where expression ϕ is satisfied: we simply associate each term τ in $\text{Sat}(\phi, X)$ with the probability $P[\tau|\phi, \Theta]$. It is easy to verify that $\sum_{\tau \in \text{Sat}(\phi, X)} P[\tau|\phi, \Theta] = 1$. Similarly, we can associate each term τ' in $\text{Dsat}(\phi, X, Y)$ with the probability $P[\tau'|\phi, \Theta]$ and obtain a proper probability distribution over $\text{Dsat}(\phi, X, Y)$. For brevity we will denote both distributions by $P[\cdot|\phi, \Theta]$.

Let ϕ by a dynamic Boolean expression and ψ be one of its almost read-once dynamic d-tree representations. In the following we will show that sampling from $\text{Dsat}(\phi, X, Y)$ w.r.t. distribution $P[\cdot|\phi, \Theta]$ takes linear time in the size of ψ. First we show that sampling from the solutions of a read-once expression takes linear time w.r.t. the size of the expression. Given a read-once expression ψ and a set of parameters Θ, Algorithm 4 (SampleReadOnceSat) returns a term expression from $\text{Sat}(\phi, \text{Var}(\phi))$, sampled w.r.t. the distribution $P[\cdot|\psi, \Theta]$. Algorithm 5 (SampleReadOnceUnsat) returns a term from $\text{Sat}(\neg\psi, \text{Var}(\psi))$, sampled w.r.t. the distribution $P[\cdot|\neg\psi, \Theta]$. Notice that the two Algorithms invoke each other at lines 19 and 21. Both Algorithms assume that all the subexpressions ψ_i appearing in the input expression ψ have been pre-annotated with their respective probabilities $P[\psi_i|\Theta]$. This computation can be done in linear time using Algorithm 3 and it is here omitted for the sake of conciseness. Lines 2–7 in Algorithm 4 are straightforward: if we need to sample a term expression from $\text{Sat}((x_i \in V), \{x_i\})$, we simply sample a value from V according to the categorical distribution $P[\cdot|x_i \in V, \Theta]$; if instead we need to sample a term from $\text{Sat}((\psi_1 \odot \psi_2), X)$, we simply sample one satisfying assignment for each expression and then merge the pair into a single assignment. To better understand lines 8–23, consider the following Proposition:

Proposition 6. *Let $\{\psi_1, \ldots, \psi_k\}$ be a collection of k mutually exclusive Boolean expressions and let ψ^* be the disjunction $\vee_{i=1}^{k} \psi_i$. For every expression ψ_j in $\{\psi_1, \ldots, \psi_k\}$ the following holds true*

$$P[\psi_j|\psi^*, \Theta] = \frac{P[\psi_j|\Theta]}{\sum_{i=1}^{k} P[\psi_i|\Theta]} \tag{11}$$

Algorithm 4: SampleReadOnceSat

Input: A read-once expression ψ, a probability distribution $P[\cdot|\Theta]$.
Output: A term-expression τ sampled from $\textsc{Sat}(\psi, X)$ with probability $P[\tau|\psi, \Theta]$.

1 *SampleReadOnceSat*(ψ, Θ):
2 **if** $\psi = (x_i \in V)$ **then**
3 **return** $(x_i = v^*)$, with v^* sampled from $P[x_i|x_i \in V, \Theta]$
4 **else if** $\psi = \psi_1 \odot \psi_2$ **then**
5 $\tau_1 \leftarrow$ SampleReadOnceSat(ψ_1, Θ)
6 $\tau_2 \leftarrow$ SampleReadOnceSat(ψ_2, Θ)
7 **return** $\tau_1 \wedge \tau_2$
8 **else if** $\psi = \psi_1 \otimes \psi_2$ **then**
9 $w_1 \leftarrow P[\psi_1|\Theta] \cdot P[\psi_2|\Theta]$
10 $w_2 \leftarrow P[\psi_1|\Theta] \cdot (1 - P[\psi_2|\Theta])$
11 $w_3 \leftarrow (1 - P[\psi_1|\Theta]) \cdot P[\psi_2|\Theta]$
12 $w_s \leftarrow w_1 + w_2 + w_3$
13 $r \leftarrow$ RndUniform(0,1)
14 **if** $r < w_1/w_s$ **then**
15 $\tau_1 \leftarrow$ SampleReadOnceSat(ψ_1, Θ)
16 $\tau_2 \leftarrow$ SampleReadOnceSat(ψ_2, Θ)
17 **else if** $r < (w_1 + w_2)/w_s$ **then**
18 $\tau_1 \leftarrow$ SampleReadOnceSat(ψ_1, Θ)
19 $\tau_2 \leftarrow$ SampleReadOnceUnsat(ψ_2, Θ)
20 **else**
21 $\tau_1 \leftarrow$ SampleReadOnceUnsat(ψ_1, Θ)
22 $\tau_2 \leftarrow$ SampleReadOnceSat(ψ_2, Θ)
23 **return** $\tau_1 \wedge \tau_2$

Algorithm 5: SampleReadOnceUnsat

Input: A read-once expression ψ, a probability distribution $P[\cdot|\Theta]$.
Output: A term-expression τ sampled from $\textsc{Sat}(\neg\psi, X)$ with probability $P[\tau|\neg\psi, \Theta]$.

1 *SampleReadOnceUnsat*(ψ, Θ):
2 **if** $\psi = (x_i \in V)$ **then**
3 **return** $(x_i = v^*)$, with v^* sampled from $P[x_i|x_i \notin V, \Theta]$
4 **else if** $\psi = \psi_1 \otimes \psi_2$ **then**
5 $\tau_1 \leftarrow$ SampleReadOnceUnsat(ψ_1, Θ)
6 $\tau_2 \leftarrow$ SampleReadOnceUnsat(ψ_2, Θ)
7 **return** $\tau_1 \wedge \tau_2$
8 **else if** $\psi = \psi_1 \odot \psi_2$ **then**
9 $w_1 \leftarrow (1 - P[\psi_1|\Theta]) \cdot (1 - P[\psi_2|\Theta])$
10 $w_2 \leftarrow (1 - P[\psi_1|\Theta]) \cdot P[\psi_2|\Theta]$
11 $w_3 \leftarrow P[\psi_1|\Theta] \cdot (1 - P[\psi_2|\Theta])$
12 $w_s \leftarrow w_1 + w_2 + w_3$
13 $r \leftarrow$ RndUniform(0,1)
14 **if** $r < w_1/w_s$ **then**
15 $\tau_1 \leftarrow$ SampleReadOnceUnsat(ψ_1, Θ)
16 $\tau_2 \leftarrow$ SampleReadOnceUnsat(ψ_2, Θ)
17 **else if** $r < (w_1 + w_2)/w_s$ **then**
18 $\tau_1 \leftarrow$ SampleReadOnceUnsat(ψ_1, Θ)
19 $\tau_2 \leftarrow$ SampleReadOnceSat(ψ_2, Θ)
20 **else**
21 $\tau_1 \leftarrow$ SampleReadOnceSat(ψ_1, Θ)
22 $\tau_2 \leftarrow$ SampleReadOnceUnsat(ψ_2, Θ)
23 **return** $\tau_1 \wedge \tau_2$

Proof.

$$P[\psi_j|\psi^*, \Theta] = \frac{P[\psi_j \wedge \psi^*|\Theta]}{P[\psi^*|\Theta]} = \frac{P[\psi_j|\Theta]}{P[\psi^*|\Theta]} = \frac{P[\psi_j|\Theta]}{\sum_{i=1}^{k} P[\psi_i|\Theta]} \quad (12)$$

□

To sample a term that satisfies $\psi_1 \otimes \psi_2$, at lines 8–23 Algorithm 4 samples one expression from the set $\{(\psi_1\psi_2), (\neg\psi_1\psi_2), (\psi_1\neg\psi_2)\}$ w.r.t. distribution $P[\cdot \mid (\psi_1 \vee \psi_2), \Theta]$ and then proceeds to sample a satisfying assignment for the selected expression. Notice that the three expressions are all mutually exclusive, and this makes it easy to compute $P[\cdot \mid (\psi_1 \vee \psi_2), \Theta]$, as per Proposition 6. Similar considerations apply to Algorithm 5.

Algorithm 6 (SampleDSat) generalizes Algorithm 4 to sample from $\textsc{Dsat}(\psi, X, Y)$, the set of terms that satisfy a dynamic d-tree expression ψ. Notice that the Algorithm takes linear time in the size of ψ and works exclusively with almost-read-once expressions, as the ones generated by Algorithm 2.

2.4 Boolean Expressions over Exchangeable Random Variables

Let x_i be a categorical variable with domain $\textsc{Dom}(x_i) = \{v_1, \ldots, v_c\}$ and $\boldsymbol{\alpha_i} = (\alpha_{i,1}, \ldots, \alpha_{i,c})$ be a c-vector of real-valued positive numbers. We denote by $P[x_i|\boldsymbol{\alpha_i}]$ the probability mass function of a Dirichlet-categorical compound distribution.

$$P[x_i|\boldsymbol{\alpha_i}] = \int_{\mathcal{S}_c} P[x_i|\boldsymbol{\theta_i}] \cdot p[\boldsymbol{\theta_i}|\boldsymbol{\alpha_i}]\, d\boldsymbol{\theta_i} \quad (13)$$

In Equation 13 the probability mass function $P[x_i|\boldsymbol{\theta_i}]$ is defined as in Equation 7, the symbol $\mathcal{S}_c$ denotes the c-dimensional probabilistic simplex (the set of all real-valued c-vectors whose components are non-negative and sum up to one), while $p[\boldsymbol{\theta_i}|\boldsymbol{\alpha_i}]$ denotes the probability density function of a Dirichlet distribution:

$$p[\boldsymbol{\theta_i}|\boldsymbol{\alpha_i}] = \frac{\prod_{j=1}^{c} \theta_{i,j}^{\alpha_{i,j}-1}}{\boldsymbol{B}(\boldsymbol{\alpha_i})} \quad (14)$$

In Equation 14 the symbol $\boldsymbol{B}(\cdot)$ denotes the *generalized Beta function*, which serves as a normalizing factor in $p[\boldsymbol{\theta_i}|\boldsymbol{\alpha_i}]$.

$$\boldsymbol{B}(\boldsymbol{\alpha_i}) = \int_{\mathcal{S}_c} \prod_{j=1}^{c} \theta_{i,j}^{\alpha_{i,j}-1}\, d\boldsymbol{\theta_i} = \frac{\prod_{j=1}^{c} \Gamma(\alpha_{i,j})}{\Gamma(\sum_{j=1}^{c} \alpha_{i,j})} \quad (15)$$

By $\Gamma(\alpha)$ we denote the standard Gamma function $\int_0^{+\infty} z^{\alpha-1} e^{-z} dz$. Under the above assumptions, we can describe x_i as a categorically-distributed random variable whose parameters are determined

Algorithm 6: SampleDSat

Input: An almost read-once dynamic d-tree expression ψ, a probability distribution $P[\cdot|\Theta]$.
Output: A term-expression τ sampled from $\textsc{Dsat}(\psi, X)$ with probability $P[\tau|\psi, \Theta]$.

1 *SampleDSat*(ψ, X, Y, Θ):
2 **if** $Y \neq \emptyset$ *and* $\psi = \oplus^{AC(y)}(\psi_1, \psi_2)$ **then**
3 $r \leftarrow$ RndUniform(0,1)
4 **if** $r < (P[\psi_1|\Theta]/(P[\psi_1|\Theta] + P[\psi_2|\Theta]))$ **then**
5 **return** SampleDSat(ψ_1, X, $Y - \{y\}$, Θ)
6 **else**
7 **return** SampleDSat(ψ_2, $X \cup \{y\}$, $Y - \{y\}$, Θ)
8 **else if** $\psi = \oplus^{x}(((x = v_1) \odot \psi_1), \ldots, ((x = v_k) \odot \psi_k))$ **then**
9 $w_s \leftarrow \sum_{i=1}^{k} P[(x = v_i) \wedge \psi_i|\Theta]$
10 Sample $v_j \in \textsc{Dom}(x)$ with prob. $P[(x = v_j) \wedge \psi_j|\Theta]/w_s$
11 **return** SampleDSat(ψ_j, X, $\emptyset$, Θ)
12 **else**
13 **return** SampleReadOnceSat(ψ, Θ)

by another random variable ($\boldsymbol{\theta}_i$) that is Dirichlet-distributed. Accordingly, we will often refer to $p[\boldsymbol{\theta}_i|\boldsymbol{\alpha}_i]$ as the *prior density* of random variable $\boldsymbol{\theta}_i$, to $P[x_i|\boldsymbol{\alpha}_i]$ as the *likelihood* of random variable x_i, and to $\boldsymbol{\alpha}_i$ as to a set of *hyper-parameters*. The integral in Equation 13 admits a closed solution, that provides a convenient way to express the likelihood $P[x_i|\boldsymbol{\alpha}_i]$ as a categorical distribution:

$$P[x_i|\boldsymbol{\alpha}_i] = \frac{\prod_{j=1}^{c} \alpha_{i,j}^{[x_i=v_j]}}{\sum_{j=1}^{c} \alpha_{i,j}} \tag{16}$$

Let $\hat{\boldsymbol{x}}_i = \{\hat{x}_i[1], \ldots, \hat{x}_i[q]\}$ be a set of q categorically-distributed random variables that take values in $\textsc{Dom}(x_i)$ and all share the same random parameters vector $\boldsymbol{\theta}_i$. By construction, vector $\hat{\boldsymbol{x}}_i$ is a random variable itself and follows a Dirichlet-multinomial compound distribution.

$$P[\hat{\boldsymbol{x}}_i|\boldsymbol{\alpha}_i] = \int_{\mathcal{S}_c} P[\hat{\boldsymbol{x}}_i|\boldsymbol{\theta}_i] \cdot p[\boldsymbol{\theta}_i|\boldsymbol{\alpha}_i]\, d\boldsymbol{\theta}_i \tag{17}$$

In Equation 17 the symbol $P[\hat{\boldsymbol{x}}_i|\boldsymbol{\theta}_i]$ represents the probability mass function of a multinomial distribution. If we denote by $n(\hat{\boldsymbol{x}}_i, v_j)$ the number of times random variable x_i takes value v_j in $\hat{\boldsymbol{x}}_i$, then $P[\hat{\boldsymbol{x}}_i|\boldsymbol{\theta}_i]$ is defined as follows:

$$P[\hat{\boldsymbol{x}}_i|\boldsymbol{\theta}_i] = \prod_{j=1}^{c} \theta_{i,j}^{n(\hat{\boldsymbol{x}}_i, v_j)} \tag{18}$$

Under the above assumptions, we say that the random variables $\{\hat{x}_i[1], \ldots, \hat{x}_i[q]\}$ are pairwise *exchangeable* [15] and *conditionally independent*. They are exchangeable because $P[\hat{\boldsymbol{x}}_i|\boldsymbol{\alpha}_i]$ is invariant w.r.t. to any permutation of the assignments to the variables in $\{\hat{x}_i[1], \ldots, \hat{x}_i[q]\}$. They are conditionally independent because they are statistically independent whenever the value of random variable $\boldsymbol{\theta}_i$ is known. In other words, for every pair of distinct variables $\hat{x}_i[n_1]$ and $\hat{x}_i[n_2]$ in $\hat{\boldsymbol{x}}_i$, $P[\hat{x}_i[n_1], \hat{x}_i[n_2]|\boldsymbol{\theta}_i] = P[\hat{x}_i[n_1]|\boldsymbol{\theta}_i] \cdot P[\hat{x}_i[n_2]|\boldsymbol{\theta}_i]$. Both properties follow immediately from Equation 18. From now on, we will refer to the variables $\{\hat{x}_i[1], \ldots, \hat{x}_i[q]\}$ as *exchangeable instances* of latent variable x_i. The likelihood of a set of instances ($P[\hat{\boldsymbol{x}}_i|\boldsymbol{\alpha}_i]$) can be expressed as follows

$$P[\hat{\boldsymbol{x}}_i|\boldsymbol{\alpha}_i] = \frac{\Gamma(\sum_{j=1}^{c} \alpha_{i,j})}{\Gamma(q + \sum_{j=1}^{c} \alpha_{i,j})} \cdot \prod_{j=1}^{c} \frac{\Gamma(\alpha_{i,j} + n(\hat{\boldsymbol{x}}_i, v_j))}{\Gamma(\alpha_{i,j})} \tag{19}$$

Notice that, by Equation 19, the probability $P[\hat{x}_i[n_1], \hat{x}_i[n_2]|\boldsymbol{\alpha}_i]$ is *not* equal to $P[\hat{x}_i[n_1]|\boldsymbol{\alpha}_i] \cdot P[\hat{x}_i[n_2]|\boldsymbol{\alpha}_i]$. In other words, when the value of variable $\boldsymbol{\theta}_i$ is not known, the random variables $\hat{x}_i[n_1]$ and $\hat{x}_i[n_2]$ are *not* statistically independent. Since the Dirichlet distribution is a conjugate prior for the multinomial distribution, the *posterior density* $p[\boldsymbol{\theta}_i|\hat{\boldsymbol{x}}_i, \boldsymbol{\alpha}_i]$ is another Dirichlet distribution:

$$p[\boldsymbol{\theta}_i|\hat{\boldsymbol{x}}_i, \boldsymbol{\alpha}_i] = \frac{P[\hat{\boldsymbol{x}}_i|\boldsymbol{\theta}_i] \cdot p[\boldsymbol{\theta}_i|\boldsymbol{\alpha}_i]}{P[\hat{\boldsymbol{x}}_i|\boldsymbol{\alpha}_i]} = \frac{\prod_{j=1}^{c} \theta_{i,j}^{\alpha_{i,j} + n(\hat{\boldsymbol{x}}_i, v_j) - 1}}{\boldsymbol{B}(\boldsymbol{\alpha}_i + \boldsymbol{n}(\hat{\boldsymbol{x}}_i))} \tag{20}$$

In Equation 20 the symbol $\boldsymbol{n}(\hat{\boldsymbol{x}}_i)$ denotes the c-vector ($n(\hat{\boldsymbol{x}}_i, v_1)$, ..., $n(\hat{\boldsymbol{x}}_i, v_c)$). The *posterior predictive* $P[x_i|\hat{\boldsymbol{x}}_i, \boldsymbol{\alpha}_i]$ takes a form similar to Equation 16

$$P[x_i|\hat{\boldsymbol{x}}_i, \boldsymbol{\alpha}_i] = \int_{\mathcal{S}_c} P[x_i|\boldsymbol{\theta}_i] \cdot p[\boldsymbol{\theta}_i|\hat{\boldsymbol{x}}_i, \boldsymbol{\alpha}_i] d\boldsymbol{\theta}_i = \frac{\prod_{j=1}^{c} (\alpha_{i,j} + n(\hat{\boldsymbol{x}}_i, v_j))^{[x_i=v_j]}}{\sum_{j=1}^{c} (\alpha_{i,j} + n(\hat{\boldsymbol{x}}_i, v_j))} \tag{21}$$

Let $X = \{x_i\}_i$ be a collection of random variables, $\boldsymbol{A} = \{\boldsymbol{\alpha}_i\}_i$ be a set of hyper-parameters associated with the variables in X, and $\hat{X} = \cup_{x_i \in X}(\hat{\boldsymbol{x}}_i)$ be a collection of exchangeable instances of the variables in X. Two distinct random variables in $\hat{X}$ are statistically independent when they refer to distinct variables in X; otherwise they are exchangeable and conditionally independent (but not fully independent). Similarly to what we did in the previous sections, we can build Boolean expressions using the random variables in $\hat{X}$ (instead of X) as literals. We call these formulas *o-expressions* (or *observed expressions*), to distinguish them from regular expressions that only refer to variables in X. Let ϕ be an o-expression. We say that ϕ is *correlation-free* if each random variable appearing in it is statistically independent from all the others. This means that each variable in X can contribute at most one instance to ϕ, and the same instance may possibly appear more than once. For example, the o-expression $(\hat{x}_1[1]\hat{x}_2[1] \vee \neg\hat{x}_1[1]\hat{x}_3[1])$ is correlation-free, but the o-expression $(\hat{x}_1[1]\neg\hat{x}_1[2])$ is not. We say that two o-expressions ϕ_1 and ϕ_2 are conditionally independent when $\textsc{Var}(\phi_1)$ and $\textsc{Var}(\phi_2)$ do not overlap. We say that they are independent when $P[\phi_1 \wedge \phi_2|\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2] = P[\phi_1|\boldsymbol{\alpha}_1] \cdot P[\phi_2|\boldsymbol{\alpha}_2]$. This happens when there are no two variable instances from ϕ_1 and ϕ_2 that refer to the same variable in X. For example, the o-expressions $(\hat{x}_1[1]\neg\hat{x}_2[1])$ and $(\hat{x}_1[2]\neg\hat{x}_2[2])$ are conditionally independent, but not fully independent; the o-expressions $(\hat{x}_1[1]\neg\hat{x}_2[1])$ and $(\hat{x}_3[1]\neg\hat{x}_4[1])$, on the other hand, are fully independent. It is easy to see that full statistical independence implies conditional independence. Let τ be a term expression in $\textsc{Asst}(\hat{X} - \textsc{Var}(\phi))$, that assigns a value to every random variable in $\hat{X}$ that does not appear in o-expression ϕ. By construction, τ and ϕ are conditionally independent. Let ψ be an almost read-once d-tree expression, obtained by applying Algorithm 2 to o-expression ϕ. If ϕ is correlation-free, then so is ψ and we can use Algorithm 6 to sample terms from $\textsc{Sat}(\phi, \textsc{Var}(\phi))$ w.r.t. conditional distribution $P[\cdot|\tau, \boldsymbol{A}]$. Similarly, we can use Algorithm 3 to compute $P[\phi|\tau, \boldsymbol{A}]$. Notice that $P[\cdot|\tau, \boldsymbol{A}]$ assigns to each random variable in $\textsc{Var}(\phi)$ a marginal likelihood that is a simple categorical distribution, as per Equation 21. Similar considerations apply to dynamic Boolean expressions.

3 GAMMA PROBABILISTIC DATABASES

The data model we introduce here is a generalization of the Dirichlet Probabilistic Databases that were originally proposed in [46].

Definition 2 (δ-tuples and δ-tables). *Let V be a finite set of two or more tuples that all share the same schema. A* δ-tuple x_i *is defined as a Dirichlet-categorical random variable, with latent parameters $\boldsymbol{\theta}_i$ and known hyper-parameters $\boldsymbol{\alpha}_i$, that takes values in V. A* δ-table *is a finite collection of pairwise independent δ-tuples, whose domains do not overlap but all share the same schema.*

Example 3.1. Figure 2 depicts two δ-tables, "Roles" (R) and "Seniority" (S), and one deterministic relation "Evidence" (E). The first δ-table consists of two δ-tuples, labeled x_1 and x_2. They encode a probabilistic hypothesis about the roles served by two employees in a fictitious company. In the example, each employee can serve in exactly one of three positions: Tech Lead, Developer or QA Engineer. The two δ-tuple assign a probability to each one of these positions. Each δ-tuple is associated with a bundle of three regular tuples, that defines its domain: the first δ-tuple (x_1) takes values in the set {(Ada, Lead), (Ada, Dev), (Ada, QA)}, the second one (x_2) takes values in the set {(Bob, Lead), (Bob, Dev), (Bob, QA)}. For convenience each value is annotated with a unique identifier ($v_{1,1}$, $v_{1,2}$, etc.). Each δ-tuple x_i is also annotated with a vector of latent parameters' identifiers $\boldsymbol{\theta}_i$, together with an assignment to the hyper-parameters $\boldsymbol{\alpha}_i$, that fully determine its distribution. When we depict a δ-table in tabular form, we report the variables' identifiers in column X, the values' identifiers in

Roles (R)

emp	role	X	V	Θ	$\boldsymbol{A}$
Ada	Lead	x_1	$v_{1,1}$	$\theta_{1,1}$	$\alpha_{1,1} = 4.1$
Ada	Dev	x_1	$v_{1,2}$	$\theta_{1,2}$	$\alpha_{1,2} = 2.2$
Ada	QA	x_1	$v_{1,3}$	$\theta_{1,3}$	$\alpha_{1,2} = 1.3$
Bob	Lead	x_2	$v_{2,1}$	$\theta_{2,1}$	$\alpha_{2,1} = 1.1$
Bob	Dev	x_2	$v_{2,2}$	$\theta_{2,2}$	$\alpha_{2,2} = 3.7$
Bob	QA	x_2	$v_{2,3}$	$\theta_{2,3}$	$\alpha_{2,3} = 0.2$

Seniority (S)

emp	exp	X	V	Θ	$\boldsymbol{A}$
Ada	Senior	x_3	$v_{3,1}$	$\theta_{3,1}$	$\alpha_{3,1} = 1.6$
Ada	Junior	x_3	$v_{3,2}$	$\theta_{3,2}$	$\alpha_{3,2} = 1.2$
Bob	Senior	x_4	$v_{4,1}$	$\theta_{4,1}$	$\alpha_{4,1} = 9.3$
Bob	Junior	x_4	$v_{4,2}$	$\theta_{4,2}$	$\alpha_{4,2} = 9.7$

Evidence (E)

role	Φ
Lead	$e1$
Dev	$e2$
QA	$e3$

Figure 2: A simple Gamma Probabilistic Database.

column V, the latent parameters in columns Θ and the hyper-parameters in column $\boldsymbol{A}$. A term expression in $\textsc{Asst}(\{x_1, x_2\})$ represents a *possible world* for δ-table "Roles", i.e. a deterministic relation with exactly two tuples. For example, the term $(x_1 = v_{1,1}) \wedge (x_2 = v_{2,2})$ identifies the possible world of δ-table "Roles" where Ada is a tech lead and Bob is a developer. Overall, δ-table "Roles" can be seen as a probability distribution over the collection of its six possible worlds. Each possible world $\tau \in \textsc{Asst}(\{x_1, x_2\})$ has probability $P[\tau|\boldsymbol{A}]$ defined as follows:

$$P[\tau|\boldsymbol{A}] = \prod_{i=1}^{n} \left(\frac{\prod_{j=1}^{c} \alpha_{i,j}^{[\tau \models (x_i = v_j)]}}{\sum_{j=1}^{c} \alpha_{i,j}} \right) \tag{22}$$

Definition 3 (Gamma Probabilistic Database). *A Gamma Probabilistic Database is a finite collection of δ-tables and regular, deterministic relations.*

The two δ-tables R and S in Figure 2, together with the deterministic table E, exemplify a well-formed Gamma Probabilistic Database. In the following, with limited abuse of notation, we use X to denote the set of all the δ-tuples in a Gamma database ($\{x_i\}_i$), V to denote the set of all the value-identifiers ($\{v_{i,j}\}_{i,j}$), Θ to denote the set of all the latent parameters ($\{\boldsymbol{\theta}_i\}_i$) and $\boldsymbol{A}$ to denote the set of all the hyper-parameters ($\{\boldsymbol{\alpha}_i\}_i$). For the sake of clarity, and without lack of generalization, we are going to assume that all the tuples in the deterministic relations are annotated with a unique identifier ($\{e_1, e_2, ...\}$).

The elements of $\textsc{Asst}(X)$ represent the *possible worlds* of a Gamma database. If ϕ is a Boolean expression with $\textsc{Var}(\phi) \subseteq X$, then the probability of sampling a possible world that satisfies ϕ is given by the following Equation

$$P[\phi|\boldsymbol{A}] = \sum_{\tau \in \textsc{Sat}(\phi, X)} P[\tau|\boldsymbol{A}] \tag{23}$$

Just like regular, deterministic databases, Gamma Probabilistic Databases support the execution of queries. We represent queries using positive set-based relational algebra operators [1]: σ_c (selection), π (projection), $\bowtie$ (natural join). A *Boolean query* is a relational algebra expression whose last operation is $\pi_{\emptyset}(\cdot)$, the projection over the empty schema $\emptyset$. If R is a deterministic relation instance, the operator $\pi_{\emptyset}(R)$ returns the value $\top$ whenever R contains at least one tuple, and the value $\bot$ when R is empty. If q is a Boolean query, we denote by $P[q|\boldsymbol{A}]$ the probability of sampling a possible world where q evaluates to $\top$. Notice that possible worlds are well-formed *deterministic* database instances, hence query evaluation is well defined. It is well-known that Boolean queries can be represented in terms of Boolean expressions encoding their *lineage* [8, 26]. Lineage expressions are built according to the following rules:

(1) the lineage of a deterministic tuple e_i is the term $e_i = \top$.
(2) the lineage of a δ-tuple value $v_{i,j}$ is the term $x_i = v_{i,j}$.
(3) if t is a tuple obtained by joining tuples t_1 and t_2, with lineage ϕ_1 and ϕ_2, then the lineage of t is $\phi_1 \wedge \phi_2$.
(4) if t is a tuple with lineage ϕ, then $\sigma_c(t)$ has lineage ϕ when t satisfies condition c, and lineage $\bot$ otherwise.
(5) if deterministic relation R contains tuples $\{t_1, \ldots, t_n\}$ with lineage expressions $\{\phi_1, \ldots, \phi_n\}$, then expression $\pi_{\emptyset}(R)$ has lineage $\vee_{i=1}^{n} \phi_i$.

Example 3.2. Let q be a Boolean query defined as follows

$$q = \pi_{\emptyset}(\sigma_{\text{role=Lead} \wedge \text{exp=Senior}}(\text{Roles} \bowtie \text{Seniority}))$$

The lineage of q is $((x_1 = v_{1,1})(x_3 = v_{3,1})) \vee ((x_2 = v_{2,1})(x_4 = v_{4,1}))$. Notice that q identifies the set of possible worlds where there is at least one senior tech lead.

If q is a Boolean query and ϕ is its lineage expression, the probability of sampling a possible world where q evaluates to $\top$ (that we denote as $P[q|\boldsymbol{A}]$) is equal to the probability of sampling a possible world where ϕ is satisfied ($P[\phi|\boldsymbol{A}]$). This means that we can use Algorithms 2 and 3 to compute $P[q|\boldsymbol{A}]$, or Algorithm 6 to sample a possible world where q evaluates to $\top$. Furthermore, we can compute the posterior density of any latent parameter $\boldsymbol{\theta}_i$ in Θ w.r.t. the observation of query q being satisfied [46]:

$$p[\boldsymbol{\theta}_i|\phi, \boldsymbol{A}] = \sum_{j=1}^{c} p[\boldsymbol{\theta}_i|(x_i = v_j), \boldsymbol{A}] \cdot P[(x_i = v_j)|\phi, \boldsymbol{A}] \tag{24}$$

By construction, the literals of a lineage expression ϕ may refer to either deterministic tuples or to δ-tuples. If all the literals in ϕ refer to deterministic tuples, then $P[\phi|\boldsymbol{A}]$ must be equal to 1 and we say that ϕ is a *deterministic* lineage expression. If at least one the literals in ϕ refers to δ-tuple, we say that ϕ is a *non-deterministic* lineage expression. Notice that all the considerations we made in Section 2.3 remain valid for lineage expressions.

From now on, for the sake of conciseness, we will identify Boolean queries with their lineage expressions (and often write ϕ in place of q). When a Boolean query ϕ represents an observed event that is used to derive a posterior density, as in Equation 24, we call ϕ a *query-answer*.

A *Belief Update* [46] is the act of re-parametrizing the probabilistic database to minimize the Kullback–Leibler (KL) divergence between the database itself and its posterior distribution w.r.t. some query-answer ϕ. The KL-divergence is defined as follows:

$$\text{KL}^{\text{div}}(\boldsymbol{A}', \boldsymbol{A}) = \sum_{i=1}^{n} \int_{\text{Dom}(\boldsymbol{\theta}_i)} p[\boldsymbol{\theta}_i|\phi, \boldsymbol{A}] \cdot \ln \left(\frac{p[\boldsymbol{\theta}_i|\phi, \boldsymbol{A}]}{p[\boldsymbol{\theta}_i|\boldsymbol{A}']} \right) d\boldsymbol{\theta}_i \tag{25}$$

In response to observing ϕ, a Belief Update replaces the old parameters $\boldsymbol{A}$ with a new set of parameters $\boldsymbol{A}^*$, defined as follows

$$\boldsymbol{A}^* = \arg\min_{\boldsymbol{A}'} \text{KL}^{\text{div}}(\boldsymbol{A}', \boldsymbol{A}) \tag{26}$$

As shown in [46], $\boldsymbol{A}^*$ can be computed by matching the sufficient statistics of $p[\Theta|\boldsymbol{A}^*]$ with the ones of $p[\Theta|\phi, \boldsymbol{A}]$. To do so, one

should pick $\boldsymbol{A}^*$ to satisfy the following set of constraints:

$$\forall \theta_{i,j} \quad \int_0^1 \ln(\theta_{i,j}) \cdot p[\theta_{i,j}|\boldsymbol{\alpha}_i^*]\, d\theta_{i,j} = \int_0^1 \ln(\theta_{i,j}) \cdot p[\theta_{i,j}|\phi, \boldsymbol{\alpha}_i]\, d\theta_{i,j} \tag{27}$$

The integrals on the left-hand side of Equation 27 admit a closed solution ($F(\alpha_{i,j}) - F(\sum_{j=1}^{c} \alpha_{i,j})$, where $F(\cdot)$ denotes the Digamma function). Furthermore, [46] shows that the expectations on the right-hand side of Equation 27 can be computed in polynomial time in the size of the database whenever ϕ is a hierarchical query [13]. Beyond hierarchical queries, computing a Bayesian update remains a #P-hard problem in the general case.

3.1 Exchangeable query-answers

The framework proposed in [46] defines Belief Updates under the assumption that all query-answers are Boolean-valued, correlation-free, and pair-wise independent. In this paper we relax the latter assumption, allowing the use of *exchangeable query-answers*, represented by Boolean expressions that are *conditionally* independent. Before doing so, we formalize the concepts of *cp*-table [63] and *o*-table. Let $\mathcal{H}$ be a Gamma probabilistic database, and $\boldsymbol{q}$ be a *non-Boolean* query. Running $\boldsymbol{q}$ against $\mathcal{H}$ produces a *cp*-table $\boldsymbol{q}(\mathcal{H})$, that is a regular relation instance where each tuple is annotated with its lineage expression [63].

Example 3.3. Let $\boldsymbol{q}$ be a non-Boolean query defined as follows

$$\boldsymbol{q} = \pi_{\text{role}}(\sigma_{\text{role}\neq\text{QA}\wedge\text{exp=Senior}}(\text{Roles} \bowtie \text{Seniority}))$$

Its execution against the Gamma probabilistic database $\mathcal{H}$ from Figure 2 produces the *cp*-table $\boldsymbol{q}(\mathcal{H})$ shown in Figure 3. Notice that the two lineage expressions in $\boldsymbol{q}(\mathcal{H})$ are *not* independent.

role	Φ
Lead	$(x_1 = v_{1,1} \wedge x_3 = v_{3,1}) \vee (x_2 = v_{2,1} \wedge x_4 = v_{4,1})$
Dev	$(x_1 = v_{1,2} \wedge x_3 = v_{3,1}) \vee (x_2 = v_{2,2} \wedge x_4 = v_{4,1})$

Figure 3: The *cp*-table generated by query $\boldsymbol{q}(\mathcal{H})$.

Both deterministic relations and δ-tuples qualify as *cp*-tables, since they have well-defined lineage expressions. Furthermore, *cp*-tables are closed w.r.t. the operators σ, π and $\bowtie$: running a query against a set of *cp*-tables returns another *cp*-table. Let's denote by $\Phi = \{\phi_1, \ldots, \phi_r\}$ the set of lineage expressions in a *cp*-table. It is easy to see that each item in Φ represents a Boolean query. Therefore, in the following we will model *cp*-tables as finite collections of Boolean queries, and identify non-Boolean queries by the lineage expressions they generate (accordingly, we will often write Φ in place of $\boldsymbol{q}$).

Let ϕ and χ be two independent lineage expressions, generated by two distinct, independent *cp*-tables. We denote by $o_\chi(\phi)$ the *o*-expression obtained by replacing in ϕ each literal in the form $(x_i \in V)$ with the expression $(\hat{x}_i[\chi] \in V)$. In other words, $o_\chi(\phi)$ replaces every random variable in ϕ with an exchangeable instance of it, defined as per Section 2.4. We say that the *o*-expression $o_\chi(\phi)$ is an *exchangeable observation* of expression ϕ, identified by χ. In order to generate exchangeable observations, we extend our query language with a new relational algebra operator, the *sampling-join* ($\bowtie\!::$).

Definition 4 (Sampling-Join, $\bowtie\!::$). *Let R_1 and R_2 be two cp-tables. The expression $(R_1 \bowtie:: R_2)$ denotes a many-to-one natural join, where the join attributes form a key for the right-hand side R_2 (hence, each tuple in R_1 can have at most one matching tuple in R_2). Let t_1 be a tuple from R_1, with lineage χ, and t_2 be a tuple from R_2, with lineage ϕ. If t_3 is a tuple in $(R_1 \bowtie:: R_2)$, obtained by joining t_1 with t_2, then its lineage is defined as $(\chi \wedge o_\chi(\phi))$. In other words, every tuple in $(R_1 \bowtie:: R_2)$ represents an exchangeable observation of a tuple from R_2.*

If χ is a deterministic lineage expression, then $(\chi \wedge o_\chi(\phi))$ is a well-formed *o*-expression, since χ has no random variables. Furthermore, if χ is an *o*-expression (generated by another sampling join), then $(\chi \wedge o_\chi(\phi))$ is a well-defined dynamic Boolean *o*-expression (as per Section 2.2), with regular variables $\textsc{Var}(\chi)$ and volatile variables $\textsc{Var}(o_\chi(\phi))$, where all the volatile variables share the same activation condition χ.

Definition 5 (*o*-table). *We define o-tables as cp-tables whose lineage formulas consist of o-expressions, that can be either regular or dynamic Boolean expressions.*

The sampling-join between a deterministic relation and a *cp*-table always generates an *o*-table. The same applies to sampling-joins between *o*-tables and *cp*-tables. Furthermore, under the assumptions stated in Properties 3 and 4, *o*-tables are closed w.r.t. the projection (π), join ($\bowtie$) and selection (σ) operators. In other words, we allow the join of two *o*-tables only when they are independent; we also allow the use of projection over an *o*-table only when it merges tuples that are mutually exclusive and have activation conditions that respect the assumptions stated in Property 4. We say that an *o*-table is *safe* when all its lineage expressions Φ are pairwise conditionally independent.

Example 3.4. Let $\boldsymbol{q}(\mathcal{H})$ be the cp-table defined in Example 3.3, and E be the deterministic relation from Figure 2. The sampling join $(E \bowtie:: \boldsymbol{q}(\mathcal{H}))$ returns the *o*-table shown in Figure 4. Notice that the two lineage expressions in $(E \bowtie:: \boldsymbol{q}(\mathcal{H}))$ are conditionally independent, therefore the *o*-table is safe.

role	Φ
Lead	$e_1 \wedge (\hat{x}_1[e_1] = v_{1,1} \wedge \hat{x}_3[e_1] = v_{3,1}) \vee ..$ $.. \vee (\hat{x}_2[e_1] = v_{2,1} \wedge \hat{x}_4[e_1] = v_{4,1})$
Dev	$e_2 \wedge (\hat{x}_1[e_2] = v_{1,2} \wedge \hat{x}_3[e_2] = v_{3,1}) \vee ..$ $.. \vee (\hat{x}_2[e_2] = v_{2,2} \wedge \hat{x}_4[e_2] = v_{4,1})$

Figure 4: The *o*-table generated by query $(E \bowtie:: \boldsymbol{q}(\mathcal{H}))$.

Let $\Phi = \{(\phi_1, X_1, Y_1), \ldots, (\phi_r, X_r, Y_r)\}$ be the set of lineage expressions generated by a safe *o*-table. Notice that the elements of Φ can be either regular or dynamic Boolean expressions. Collectively, they identify a set of possible worlds $\mathcal{W}$ for the underlying database: $\mathcal{W} = \{\bigwedge_{i=1}^{r} \tau_i | (\tau_1, .., \tau_r) \in (\textsc{Dsat}(\phi_1, X_1, Y_1) \times \ldots \times \textsc{Dsat}(\phi_r, X_r, Y_r))\}$. While we cannot use Algorithm 6 to sample directly from $\mathcal{W}$ w.r.t. the conditional distribution $P[\cdot|\Phi, \boldsymbol{A}]$, it is straightforward to design a Gibbs sampler [23] to perform approximate sampling over $\mathcal{W}$. To do so, we model each expression ϕ_i in Φ as a random variable that takes values in $\textsc{Dsat}(\phi_i, X_i, Y_i)$. We initialize the sampler by assigning to each ϕ_i a satisfying assignment τ_i from $\textsc{Dsat}(\phi_i, X_i, Y_i)$, chosen at random. We denote by w_0 the conjunction of all these satisfying terms. Notice that w_0 is a term-expressions in $\mathcal{W}$. In order to sample a new possible world w.r.t. $P[\cdot|\Phi, \boldsymbol{A}]$ we build a Markov chain of possible worlds, having $P[\cdot|\Phi, \boldsymbol{A}]$ as stationary distribution. If w_n is some arbitrary possible world in $\mathcal{W}$, its successor w_{n+1} in the chain is obtained by choosing at random one expression ϕ_i

in Φ, and re-assigning a satisfying term to it, sampled from the conditional distribution $P[\cdot|w_n^{-i}, \boldsymbol{A}]$. The symbol w_n^{-i} denotes the conjunction of all the terms in w_n with the exclusion of τ_i, the one assigned to expression ϕ_i. As discussed in Section 2.4, we can use Algorithm 6 to sample a term from $\textsc{Dsat}(\phi_i, X_i, Y_i)$ w.r.t. distribution $P[\cdot|w_n^{-i}, \boldsymbol{A}]$.

Proposition 7. *The probability of transitioning from possible world w_n to possible world w_{n+1} is equal the probability of transitioning from w_{n+1} to w_n. Thus, the proposed Markov chain of possible worlds is* reversible.

Proof. Let's denote by τ_i and τ_i' the terms assigned to expression ϕ_i by possible worlds w_n and w_{n+1}, respectively. The probability of transitioning from w_n to w_{n+1} is given by $(P[\tau_i'|w_n^{-i}, \boldsymbol{A}] \cdot P[w_n|\boldsymbol{A}])$. Similarly, the probability of transitioning from w_{n+1} to w_n is given by $(P[\tau_i|w_{n+1}^{-i}, \boldsymbol{A}] \cdot P[w_{n+1}|\boldsymbol{A}])$. Since w_n and w_{n+1} only differs in the value assigned to ϕ_i, we can infer that $w_n^{-i} = w_{n+1}^{-i}$. Furthermore $P[w_n|\boldsymbol{A}] = (P[\tau_i|w_n^{-i}, \boldsymbol{A}] \cdot P[w_n^{-i}, \boldsymbol{A}])$ and $P[w_{n+1}|\boldsymbol{A}] = (P[\tau_i'|w_n^{-i}, \boldsymbol{A}] \cdot P[w_n^{-i}, \boldsymbol{A}])$. The thesis follows immediately. □

Since it is always possible to transition between two arbitrary possible worlds in $\mathcal{W}$ in a finite number of steps, the proposed Markov chain is *irreducible*. Furthermore, the chain is *aperiodic*, since the choice of the expression ϕ_i to be used in a transition is made at random.

If Φ is a set of lineage expressions generated by a safe o-table, we can use the proposed Gibbs sampling strategy to perform an approximate Belief Update w.r.t. the event of observing all the expressions in Φ being satisfied. Such Belief Update should minimize the KL-divergence between the database and the posterior distribution $P[\Theta|\Phi, \boldsymbol{A}]$. As discussed in Section 3, the parameter-vector $\boldsymbol{A}^*$ that minimizes the KL-divergence w.r.t. the posterior distribution is the one that satisfies the following set of constraints:

$$\forall \theta_{i,j} \quad \int_0^1 \ln(\theta_{i,j}) \cdot p[\theta_{i,j}|\boldsymbol{\alpha}_i^*]\, d\theta_{i,j} = \int_0^1 \ln(\theta_{i,j}) \cdot p[\theta_{i,j}|\Phi, \boldsymbol{\alpha}_i]\, d\theta_{i,j} \tag{28}$$

Let $\hat{\mathcal{W}}$ be a finite collection of possible worlds that satisfy all the expressions in Φ, sampled from $P[\cdot|\Phi, \boldsymbol{A}]$ with the Gibbs method described above. In order to approximate $\boldsymbol{A}^*$, we can approximate the expectations on the right-hand side of Equation 28 as follows:

$$\int_0^1 \ln(\theta_{i,j}) p[\theta_{i,j}|\Phi, \boldsymbol{\alpha}_i]\, d\theta_{i,j} \approx \frac{1}{|\hat{\mathcal{W}}|} \sum_{\hat{w}_i \in \hat{\mathcal{W}}} \int_0^1 \ln(\theta_{i,j}) p[\theta_{i,j}|\hat{w}_i, \boldsymbol{A}]\, d\theta_{i,j} \tag{29}$$

Since $p[\theta_{i,j}|\hat{w}_i, \boldsymbol{A}]$ is a Dirichlet density function, the integrals on the right-hand side of Equation 29 always admit a closed solution (the same closed solution that we identified for the left-hand side of Equation 27).

3.2 Latent Dirichlet Allocation with Query-Answers

In this Section we show how to express the Latent Dirichlet Allocation model (or LDA [6]) in terms of query-answers. Let's assume that the text corpus is stored in a deterministic relation called "Corpus" (C) with three attributes, *dID*, *ps* and *wID*, that represent a document-identifier, a positional index, and a word-identifier, respectively. If there are D documents of average length L, then relation C has $(D \cdot L)$ records. Let W denote the size of the vocabulary (i.e. the number of distinct values in column *wID* in C). We model each topic as a δ-tuple with cardinality W and parameters' vector $\boldsymbol{\beta}$, stored in a δ-table called "Topics" (T). If there are a total of K topics, then relation T consists of K δ-tuples, for a total of $(K \cdot W)$ tuples. We assume that all the hyper-parameters $\boldsymbol{\beta}$ in T are set to the same fixed value β^*, imposing a symmetric Dirichlet prior over the topics' definition. Finally, we use an additional δ-table "Documents" (D) to store the composition of each document in terms of topics. Each δ-tuple in D represents a document, has cardinality K (the number of topics) and parameters' vector $\boldsymbol{\alpha}$. Furthermore, we assume that all the hyper-parameters $\boldsymbol{\alpha}$ in D are set to the same fixed value α^*. Figure 5 exemplifies our proposed schema. For the sake of brevity, the Figure does not include the latent parameters $\boldsymbol{\Theta}$, nor the assignment to the hyper-parameters $\boldsymbol{A}$.

Corpus (C)

dID	ps	wID	Φ
D_1	1	The	$e_{1,1}$
D_1	2	Cat	$e_{1,2}$
D_1	3	Naps	$e_{1,3}$
...	...	...	...
D_2	1	Once	$e_{2,1}$
D_2	2	Upon	$e_{2,2}$
...	...	...	...
D_D	L	End	$e_{D,L}$

Topics (T)

tID	wID	Φ	$\boldsymbol{A}$
T_1	Abate	$b_1 = v_1$	$\beta_{1,1}$
T_1	Abdicate	$b_1 = v_2$	$\beta_{1,2}$
...	...	...	...
T_1	Zealous	$b_1 = v_W$	$\beta_{1,w}$
...	...	...	...
T_k	Abate	$b_k = v_1$	$\beta_{k,1}$
...	...	...	...
T_k	Zealous	$b_k = v_W$	$\beta_{k,w}$

Documents (D)

dID	tID	Φ	$\boldsymbol{A}$
D_1	T_1	$a_1 = t_1$	$\alpha_{1,1}$
D_1	T_2	$a_1 = t_2$	$\alpha_{1,2}$
...	...	...	...
D_1	T_K	$a_1 = t_K$	$\alpha_{1,K}$
...	...	...	...
D_D	T_1	$a_D = t_1$	$\alpha_{D,1}$
...	...	...	...
D_D	T_K	$a_D = t_K$	$\alpha_{D,K}$

Figure 5: A Gamma Probabilistic Database for Latent Dirichlet Allocation.

The Latent Dirichlet Allocation model can be formulated by the following non-Boolean query:

$$\boldsymbol{q}_{\text{lda}} = \pi_{\text{dID, ps, wID}}\left((C \bowtie:: D) \bowtie:: T\right) \tag{30}$$

Let $q = (d, p, w)$ be an arbitrary tuple in $\boldsymbol{q}_{\text{lda}}(\mathcal{H})$, that identifies document a_d, position $e_{d,p}$ and word v_w. The lineage of tuple q is

$$\phi_{d,p,w} = \bigvee_{i=1}^{K} \left[(a_d[e_{d,p}] = t_i) \wedge (b_i[(a_d[e_{d,p}] = t_i)] = v_w)\right] \tag{31}$$

This lineage expression simply states that the word v_w, observed at position $e_{d,p}$ in document a_d, must have been generated by exactly one of the K topics defined in relation T. Notice that $\phi_{d,p,w}$ is a dynamic Boolean expression: the number of observed instances associated with random variable b_i (that represents the number of words observed for the i-th topic) varies dynamically with the values assigned to variable $a_d[e_{d,p}]$. Since $\boldsymbol{q}_{\text{lda}}(\mathcal{H})$ has exactly one tuple for each document d and position p, it is easy to see that $\boldsymbol{q}_{\text{lda}}$ generates a safe o-table. If we apply the Gibbs sampling strategy discussed in Section 3.1 to $\boldsymbol{q}_{\text{lda}}(\mathcal{H})$, the resulting sampling algorithm is functionally equivalent to the popular

collapsed Gibbs sampler proposed by [27]. It is sufficient to compute a Belief Update w.r.t. δ-tables T and D, as per Equation 28, to obtain the definitions of the topics and their allocation over the text corpus. It is important to point out that $\boldsymbol{q}_{\text{lda}}(\mathcal{H})$ only states the statistical assumptions of the LDA model, without providing implementation details for any specific inference method. We believe that this clear separation between probabilistic models and inference algorithms is a very desirable property of the framework, that can ease many of the data management problems that arise with uncertain data. In this paper we decided to focus on the translation of safe o-tables into Gibbs samplers, but it would be plausible to support alternative inference methods, such as Variational Inference [5] or Geometric Inference [71]. We leave these directions open for future investigation.

4 EXPERIMENTS

In this section we present experimental results to validate our framework in terms of (i) correctness and (ii) expressive power.

Correctness. In the first experiment we use the Latent Dirichlet Allocation model, in its formulation discussed in Section 3.2, to verify the soundness of our approach in a quantitative way. LDA is fairly popular, and a plethora of well-established implementations and datasets are available for it. For our experiments we chose to compare our prototype against Mallet [44], a general-purpose NLP toolkit that includes a highly-optimized implementation of the collapsed Gibbs sampler originally proposed by [27]. The two implementations are evaluated against two large text collections published[1] by the UCI Machine Learning Repository [17]: NYTIMES and PUBMED. The NYTIMES dataset consists of D=299,752 news articles, that collectively contain about 100 million tokens, with a vocabulary of W=102,660 distinct words. The PUBMED dataset consists of D=8,200,000 research paper abstracts, for a total of about 730,000,000 tokens, over a vocabulary of W=141,043 distinct words. On both datasets we hold-out 10% of the documents, selected at random, for testing and use the remaining documents for training the model. For both datasets we set the number of topics (K) to 20, the symmetric Dirichlet prior over the documents' composition (α^*) to 0.2 and the symmetric Dirichlet prior over the topics' composition (β^*) to 0.1. In Figures 6a and 6b we plot the perplexity of both the training- and the test-set against the progress of the Gibbs sampler. The perplexity over the training-set (Figure 6a) quantifies the ability of the model to fit the training data. The perplexity over the test-set (Figure 6b) quantifies the ability of the model to classify unseen data. In both cases lower values of perplexity represent better performance. Deriving the exact perplexity for LDA requires to iterate over all the possible assignments of the model's latent variables, making the computation intractable [49]. To perform our experiment we approximate the perplexity using the empirical likelihood estimator provided with Mallet (`evaluate-topics`); the technical details of the estimator are discussed in [68]. Since we use the very same estimator to evaluate both our prototype and Mallet's implementation of LDA, our comparison is fair and unbiased.

From this experiment we can draw two conclusions: First, the two implementations of LDA under exam are comparable in performance, both in terms of predictive power and bare computational efficiency. Second, this experiment highlights the impact that dynamic Boolean expressions have on the performance of our framework. As we mentioned in Section 3.2, the second sampling-join in Equation 30 generates the *dynamic* term $(b_i[(a_d[e_{d,p}] = t_i)] = v_w)$ in the lineage expression of Equation 31. In practical terms, this means that the total number of exchangeable observations for all the random variables in the set $\{b_i\}_i$ is equal to $(D \cdot L)$, the total number of tokens in the dataset (or, equivalently, the total number of records in relation "Corpus" from Section 3.2). This mirrors the Gibbs sampler designed in [27]. To appreciate the advantage of this approach, let's consider the following alternate formulation of LDA, that avoids the use of dynamically-allocated random variables, and therefore the use of any dynamic Boolean expression:

$$\boldsymbol{q}'_{\text{lda}} = \pi_{\text{dID, ps, wID}} \left(C \bowtie:: (D \bowtie T)\right) \tag{32}$$

Relations C, D and T are defined as in Section 3.2. Let $q' = (d, p, w)$ be an arbitrary tuple in $\boldsymbol{q}'_{\text{lda}}(\mathcal{H})$, that identifies document a_d, position $e_{d,p}$ and word v_w. By design, the lineage of tuple q' is free of any dynamic term:

$$\phi'_{d,p,w} = \bigvee_{i=1}^{K} \left[(a_d[e_{d,p}] = t_i) \wedge (b_i[e_{d,p}] = v_w)\right] \tag{33}$$

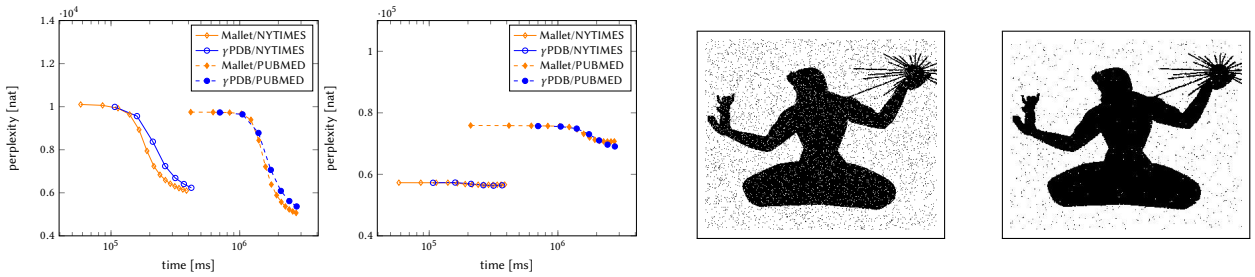
In practical terms, this means that total number of exchangeable observations for the variables in $\{b_i\}_i$ is equal to $(K \cdot D \cdot L)$. While this over-abundance of latent variables in $\boldsymbol{q}'_{\text{lda}}$ does not prevent the model from learning meaningful topics, the time required for training is increased by a factor proportional to K. In our experiments we observed a performance degradation factor of 10.46x. In conclusion, this first experiment validates our prototype in terms of correctness and shows that dynamic Boolean expressions are essential to make our framework competitive with highly-optimized tools like Mallet.

Expressive power. Unlike Mallet, our framework is designed to support a variety of Bayesian models, beyond LDA. To prove its expressive power, in our second experiment we formulate the Ising model [41] in terms of query-answers. The Ising model was originally proposed in the field of statistical mechanics as a mathematical model of ferromagnetism. It defines a probability distribution over a lattice of binary random variables, that take values in the set $\{-1, 1\}$. We call these variables *sites* and we use the lineage expression $s_{x,y}$ to identify each site by its position (x, y) in the the lattice. The Ising model postulates that the value taken by each site is influenced by two factors: First, the action of an external magnetic field, that determines the prior probability of each site taking value $+1$ or -1; second, the action of ferromagnetic interactions that tend to align contiguous sites to the same value. This model has been used as a simple tool for image denoising [41]. In this context the lattice of sites represents a bitmap, the external field represents the evidence (i.e. a black-and-white image contaminated with some high-frequency white noise), and the ferromagnetic interaction acts as a smoothing filter that separates the noise from the ground truth (i.e. the original image). In our framework an image can be represented with a δ-table having one binary δ-tuple per site:

Image (I)

x	**y**	**v**	Φ	$\boldsymbol{A}$
0	0	+1	$s_{0,0} = +1$	$\alpha_{0,0,1}$
0	0	−1	$s_{0,0} = -1$	$\alpha_{0,0,-1}$
...	...	...	...	...
$x_{\max}$	$y_{\max}$	+1	$s_{x_{\max},y_{\max}} = +1$	$\alpha_{x_{\max},y_{\max},1}$
$x_{\max}$	$y_{\max}$	−1	$s_{x_{\max},y_{\max}} = -1$	$\alpha_{x_{\max},y_{\max},-1}$

[1] https://archive.ics.uci.edu/ml/datasets/bag+of+words



(a) Training-set perplexity (b) Test-set perplexity (c) Noisy bitmap (d) MAP estimation

Figure 6

We denote by $x_{\max}$ and $y_{\max}$ the number of rows and columns in the lattice. The set of hyper-parameters $\{\alpha_{x,y,v}\}_{x,y,v}$ encode the noisy input-image. We use a collection of exchangeable query-answer to encode the effects of the ferromagnetic interactions. Let's define the two deterministic relations $L_1(x_1, y_1)$ and $L_2(x_2, y_2)$, all containing one record per each location in the lattice. We can encode the ferromagnetic interactions between each site and its right neighbor using the following query-answer:

$$V_1(x_1, y_1, v) := \pi_{x_1,y_1,v}(\sigma_{x_1=x \wedge y_1=y}(L_1 \bowtie :: I))$$
$$V_2(x_2, y_2, v) := \pi_{x_2,y_2,v}(\sigma_{x_2=x \wedge y_2=y}(L_2 \bowtie :: I))$$
$$\mathbf{q} := \pi_{x_1,y_1}(\sigma_{(x_1=x_2) \wedge y_2=(y_1+1)}(V_1 \bowtie V_2))$$

Similar query-answers can be used to encode the interactions with the other three neighbors. Figures 6c and 6d exemplify the use of the Ising model to clean up a black-and-white image. The evidence (Figure 6c) is obtained by flipping each bit in the original image with a probability of 0.05. The image in Figure 6d is obtained by performing a maximum a posteriori estimation of the model. We used the prior $\alpha_{x,y} = (3, 0)$ for the black pixels and $\alpha_{x,y} = (0, 3)$ for the white ones. This second experiment highlights our framework ability to express general probabilistic programs in terms of query-answers. Section 8 of [46] offers additional examples.

5 RELATED WORK

We start our review of the related works from the classic probabilistic programming languages like Stan [10], PyMC3 [57], Edward [65], or Pyro [4]. With respect to our framework, these languages are more mature and offer greater flexibility in the data model, supporting both continuous and discrete random variables and a wide range of probability distributions. In terms of inference, they often rely on general-purpose MCMC samplers like [18] and [31], or black-box variational inference [39, 54]. Unlike our framework, generic probabilistic programming languages are designed to access the training data in the form of flat files or array-shaped in-memory data structures from environments like R or Numpy, remaining mostly disconnected from existing data management systems and entirely oblivious to the underlying structure of the data.

The database research community has been very active on improving the support for machine learning (ML) workloads. One line of work focuses on "factorized learning", a method that exploits join-level optimizations to speed-up common gradient-descent ML tasks, expressed as relational queries. Factorized learning has been successfully used to optimize linear models [11, 40, 58], polynomial regression, ridge regression, factorization machines and decision trees [37, 59]. For future work, we see factorized learning systems as a candidate platform to execute variational inference for our framework. Monte Carlo methods have found many successful applications in the context of probabilistic databases. Good examples include the Karp-Luby approximation algorithms for model counting [35, 36], the MC-SAT slice sampler for relational domains proposed by [53], the factorized Monte-Carlo database proposed in [69] and, more recently, the simulation-oriented distributed systems like simSQL [9] and BUDS [22]. Systems like [69] do not offer the same flexibility as generic probabilistic programming, since the factorization of the model is encoded in the database itself and not in the queries. Similar considerations apply to simSQL and BUDS; while these systems can be used to implement Gibbs samplers, the actual code of the sampler must be provided by the end-user. For example, to support distributed execution the authors of [9] settled for the implementation of an uncollapsed LDA sampler. Another line of work from the database community focuses on integrating the support for array-shaped data into the standard relational model, either for scientific data [34, 62], or for matrix- and tensor-shaped data generated by ML workloads [42, 70]. All these systems are fully relational at heart, and therefore compatible with our approach. The authors of [2] propose a probabilistic programming framework based on Datalog, that supports both recursion and continuous distributions. Unlike our work, no inference method is provided. The authors of [50] show how finite exchangeability can be used to support tractable, lifted inference.

6 CONCLUSIONS AND FUTURE WORK

We introduced a novel probabilistic programming framework, where probabilistic programs are expressed as sets of exchangeable query-answers. Our system can encode non-trivial models like Latent Dirichlet Allocation, and derive a functional collapsed Gibbs sampler for it. We plan to expand our work in several directions: First, we will investigate the use of alternative inference methods, like variational [5] and geometric inference [71]. Second, we plan to integrate our framework into a proper database system, as done in [45, 46]. Third, we want to extend the framework to allow the explicit modeling of causality [55, 56], with distinct operators for conditioning and intervention, and explore the use of continuous variables, in the spirit of [29] and [28].

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley. http://webdam.inria.fr/Alice/
[2] Vince Bárány, Balder ten Cate, Benny Kimelfeld, Dan Olteanu, and Zografoula Vagena. 2017. Declarative Probabilistic Programming with Datalog. *ACM Trans. Database Syst.* 42, 4 (2017), 22:1–22:35. https://doi.org/10.1145/3132700
[3] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* 59, 1 (2017), 65–98.

https://doi.org/10.1137/141000671
[4] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. 2019. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research* 20, 1 (2019), 973–978.
[5] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. 2016. Variational Inference: A Review for Statisticians. *CoRR* abs/1601.00670 (2016). arXiv:1601.00670 http://arxiv.org/abs/1601.00670
[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (2003), 993–1022. http://jmlr.org/papers/v3/blei03a.html
[7] Matthias Boehm, Arun Kumar, and Jun Yang. 2019. *Data Management in Machine Learning Systems.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00895ED1V01Y201901DTM057
[8] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. 2001. Why and Where: A Characterization of Data Provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings (Lecture Notes in Computer Science)*, Jan Van den Bussche and Victor Vianu (Eds.), Vol. 1973. Springer, 316–330. https://doi.org/10.1007/3-540-44503-X_20
[9] Zhuhua Cai, Zografoula Vagena, Luis Leopoldo Perez, Subramanian Arumugam, Peter J. Haas, and Christopher M. Jermaine. 2013. Simulation of database-valued markov chains using SimSQL. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.). ACM, 637–648. https://doi.org/10.1145/2463676.2465283
[10] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A probabilistic programming language. *Journal of statistical software* 76, 1 (2017).
[11] Lingjiao Chen, Arun Kumar, Jeffrey F. Naughton, and Jignesh M. Patel. 2017. Towards Linear Algebra over Normalized Data. *Proc. VLDB Endow.* 10, 11 (2017), 1214–1225. https://doi.org/10.14778/3137628.3137633
[12] Yves Crama and Peter L. Hammer. 2011. *Boolean Functions - Theory, Algorithms, and Applications.* Encyclopedia of mathematics and its applications, Vol. 142. Cambridge University Press. http://www.cambridge.org/gb/knowledge/isbn/item6222210/?site_locale=en_GB
[13] Nilesh N. Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6 (2012), 30:1–30:87. https://doi.org/10.1145/2395116.2395119
[14] Adnan Darwiche and Pierre Marquis. 2011. A Knowledge Compilation Map. *CoRR* abs/1106.1819 (2011). arXiv:1106.1819 http://arxiv.org/abs/1106.1819
[15] Bruno De Finetti. 1974. Theory of probability: a critical introductory treatment. (1974).
[16] Maarten Van den Heuvel, Peter Ivanov, Wolfgang Gatterbauer, Floris Geerts, and Martin Theobald. 2019. Anytime Approximation in Probabilistic Databases via Scaled Dissociations. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1295–1312. https://doi.org/10.1145/3299869.3319900
[17] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[18] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. 1987. Hybrid monte carlo. *Physics letters B* 195, 2 (1987), 216–222.
[19] Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. 2019. Uncertainty Annotated Databases - A Lightweight Approach for Approximating Certain Answers. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1313–1330. https://doi.org/10.1145/3299869.3319887
[20] Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime approximation in probabilistic databases. *VLDB J.* 22, 6 (2013), 823–848. https://doi.org/10.1007/s00778-013-0310-5
[21] George Fishman. 2013. *Monte Carlo: concepts, algorithms, and applications.* Springer Science & Business Media.
[22] Zekai J. Gao, Shangyu Luo, Luis Leopoldo Perez, and Chris Jermaine. 2017. The BUDS Language for Distributed Bayesian Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 961–976. https://doi.org/10.1145/3035918.3035937
[23] Stuart German and Donald German. 1988. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. In *Neurocomputing: foundations of research.* 611–634.
[24] Martin Charles Golumbic and Vladimir Gurvich. 2009. Read-once functions. *Boolean Functions: Theory, Algorithms and Applications* (2009), 519–560.
[25] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, David A. McAllester and Petri Myllymäki (Eds.). AUAI Press, 220–229. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1346&proceeding_id=24
[26] Todd J. Green, Gregory Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, Leonid Libkin (Ed.). ACM, 31–40. https://doi.org/10.1145/1265530.1265535
[27] Thomas L Griffiths and Mark Steyvers. 2004. Finding scientific topics. *Proceedings of the National academy of Sciences* 101, suppl 1 (2004), 5228–5235.
[28] 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 2020, Portland, OR, USA, June 14-19, 2020*, Dan Suciu, Yufei Tao, and Zhewei Wei (Eds.). ACM, 347–360. https://doi.org/10.1145/3375395.3387659
[29] Martin Grohe and Peter Lindner. 2020. Infinite Probabilistic Databases. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark (LIPIcs)*, Carsten Lutz and Jean Christoph Jung (Eds.), Vol. 155. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 16:1–16:20. https://doi.org/10.4230/LIPIcs.ICDT.2020.16
[30] Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nat.* 585 (2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2
[31] Matthew D. Hoffman and Andrew Gelman. 2014. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.* 15, 1 (2014), 1593–1623. http://dl.acm.org/citation.cfm?id=2638586
[32] Ross Ihaka and Robert Gentleman. 1996. R: a language for data analysis and graphics. *Journal of computational and graphical statistics* 5, 3 (1996), 299–314.
[33] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J. Gao. 2020. Declarative Recursive Computation on an RDBMS: or, Why You Should Use a Database For Distributed Machine Learning. *SIGMOD Rec.* 49, 1 (2020), 43–50. https://doi.org/10.1145/3422648.3422659
[34] Alexander Kalinin, Ugur Çetintemel, and Stanley B. Zdonik. 2015. Searchlight: Enabling Integrated Search and Exploration over Large Multidimensional Data. *Proc. VLDB Endow.* 8, 10 (2015), 1094–1105. https://doi.org/10.14778/2794367.2794378
[35] Richard M. Karp and Michael Luby. 1983. Monte-Carlo Algorithms for Enumeration and Reliability Problems. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983.* IEEE Computer Society, 56–64. https://doi.org/10.1109/SFCS.1983.35
[36] Richard M. Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo Approximation Algorithms for Enumeration Problems. *J. Algorithms* 10, 3 (1989), 429–448. https://doi.org/10.1016/0196-6774(89)90038-2
[37] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2018. AC/DC: In-Database Learning Thunderstruck. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, DEEM@SIGMOD 2018, Houston, TX, USA, June 15, 2018*, Sebastian Schelter, Stephan Seufert, and Arun Kumar (Eds.). ACM, 8:1–8:10. https://doi.org/10.1145/3209889.3209896
[38] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2016. FAQ: Questions Asked Frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Tova Milo and Wang-Chiew Tan (Eds.). ACM, 13–28. https://doi.org/10.1145/2902251.2902280
[39] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2015. Automatic Variational Inference in Stan. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett (Eds.). 568–576. https://proceedings.neurips.cc/paper/2015/hash/352fe25daf686bdb4edca223c921acea-Abstract.html
[40] Arun Kumar, Jeffrey F. Naughton, and Jignesh M. Patel. 2015. Learning Generalized Linear Models Over Normalized Data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1969–1984. https://doi.org/10.1145/2723372.2723713
[41] Stan Z Li. 2009. *Markov random field modeling in image analysis.* Springer Science & Business Media.
[42] Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, and Christopher M. Jermaine. 2019. Scalable Linear Algebra on a Relational Database System. *IEEE Trans. Knowl. Data Eng.* 31, 7 (2019), 1224–1238. https://doi.org/10.1109/TKDE.2018.2827988
[43] Andrew McCallum, Karl Schultz, and Sameer Singh. 2009. FACTORIE: Probabilistic Programming via Imperatively Defined Factor Graphs. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*, Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, and Aron Culotta (Eds.). Curran Associates, Inc., 1249–1257. http://papers.nips.cc/paper/3654-factorie-probabilistic-programming-via-imperatively-defined-factor-graphs
[44] Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit. *http://mallet. cs. umass. edu* (2002).

[45] Niccolò Meneghetti, Oliver Kennedy, and Wolfgang Gatterbauer. 2017. Beta Probabilistic Databases: A Scalable Approach to Belief Updating and Parameter Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 573–586. https://doi.org/10.1145/3035918.3064026

[46] Niccolò Meneghetti, Oliver Kennedy, and Wolfgang Gatterbauer. 2018. Learning From Query-Answers: A Scalable Approach to Belief Updating and Parameter Learning. *ACM Trans. Database Syst.* 43, 4 (2018), 17:1–17:41. https://doi.org/10.1145/3277503

[47] Brian Milch, Bhaskara Marthi, Stuart J. Russell, David A. Sontag, Daniel L. Ong, and Andrey Kolobov. 2005. BLOG: Probabilistic Models with Unknown Objects. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, Leslie Pack Kaelbling and Alessandro Saffiotti (Eds.). Professional Book Center, 1352–1359. http://ijcai.org/Proceedings/05/Papers/1546.pdf

[48] Kevin Patrick Murphy. 2002. *Dynamic bayesian networks: representation, inference and learning*. University of California, Berkeley.

[49] Iain Murray and Ruslan Salakhutdinov. 2008. Evaluating probabilities under high-dimensional latent variable models. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou (Eds.). Curran Associates, Inc., 1137–1144. https://proceedings.neurips.cc/paper/2008/hash/0d7de1aca9299fe63f3e0041f02638a3-Abstract.html

[50] Mathias Niepert and Guy Van den Broeck. 2014. Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, Carla E. Brodley and Peter Stone (Eds.). AAAI Press, 2467–2475. http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8615

[51] Dan Olteanu. 2020. The Relational Data Borg is Learning. *Proc. VLDB Endow.* 13, 12 (2020), 3502–3515. https://doi.org/10.14778/3415478.3415572

[52] Dan Olteanu and Maximilian Schleich. 2016. Factorized Databases. *SIGMOD Rec.* 45, 2 (2016), 5–16. https://doi.org/10.1145/3003665.3003667

[53] Hoifung Poon and Pedro M. Domingos. 2006. Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. AAAI Press, 458–463. http://www.aaai.org/Library/AAAI/2006/aaai06-073.php

[54] Rajesh Ranganath, Sean Gerrish, and David M. Blei. 2014. Black Box Variational Inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014 (JMLR Workshop and Conference Proceedings)*, Vol. 33. JMLR.org, 814–822. http://proceedings.mlr.press/v33/ranganath14.html

[55] Babak Salimi, Leopoldo E. Bertossi, Dan Suciu, and Guy Van den Broeck. 2016. Quantifying Causal Effects on Query Answering in Databases. In *8th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2016, Washington, D.C., USA, June 8-9, 2016*, Sarah Cohen Boulakia (Ed.). USENIX Association. https://www.usenix.org/conference/tapp16/workshop-program/presentation/salimi

[56] Babak Salimi, Harsh Parikh, Moe Kayali, Lise Getoor, Sudeepa Roy, and Dan Suciu. 2020. Causal Relational Learning. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 241–256. https://doi.org/10.1145/3318464.3389759

[57] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. 2016. Probabilistic programming in Python using PyMC3. (2016).

[58] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. 2016. Learning Linear Regression Models over Factorized Joins. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 3–18. https://doi.org/10.1145/2882903.2882939

[59] Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q. Ngo, and XuanLong Nguyen. 2019. A Layered Aggregate Engine for Analytics Workloads. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1642–1659. https://doi.org/10.1145/3299869.3324961

[60] Claude E. Shannon. 1949. The synthesis of two-terminal switching circuits. *Bell Syst. Tech. J.* 28, 1 (1949), 59–98. https://doi.org/10.1002/j.1538-7305.1949.tb03624.x

[61] David J Spiegelhalter, Andrew Thomas, Nicky G Best, Wally Gilks, and D Lunn. 1996. BUGS: Bayesian inference using Gibbs sampling. *Version 0.5,(version ii) http://www. mrc-bsu. cam. ac. uk/bugs* 19 (1996).

[62] Michael Stonebraker, Paul Brown, Donghui Zhang, and Jacek Becla. 2013. SciDB: A Database Management System for Applications with Complex Analytics. *Comput. Sci. Eng.* 15, 3 (2013), 54–62. https://doi.org/10.1109/MCSE.2013.19

[63] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00362ED1V01Y201105DTM016

[64] David Tolpin, Jan-Willem van de Meent, and Frank D. Wood. 2015. Probabilistic Programming in Anglican. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III (Lecture Notes in Computer Science)*, Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavaldà, Dino Pedreschi, Francesco Bonchi, Jaime S. Cardoso, and Myra Spiliopoulou (Eds.), Vol. 9286. Springer, 308–311. https://doi.org/10.1007/978-3-319-23461-8_36

[65] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja R. Rudolph, Dawen Liang, and David M. Blei. 2016. Edward: A library for probabilistic modeling, inference, and criticism. *CoRR* abs/1610.09787 (2016). arXiv:1610.09787 http://arxiv.org/abs/1610.09787

[66] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421. https://doi.org/10.1137/0208032

[67] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *stat* 1050 (2018), 27.

[68] Hanna M. Wallach, Iain Murray, Ruslan Salakhutdinov, and David M. Mimno. 2009. Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009 (ACM International Conference Proceeding Series)*, Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman (Eds.), Vol. 382. ACM, 1105–1112. https://doi.org/10.1145/1553374.1553515

[69] Michael L. Wick, Andrew McCallum, and Gerome Miklau. 2010. Scalable Probabilistic Databases with Factor Graphs and MCMC. *Proc. VLDB Endow.* 3, 1 (2010), 794–804. https://doi.org/10.14778/1920841.1920942

[70] Binhang Yuan, Dimitrije Jankov, Jia Zou, Yuxin Tang, Daniel Bourgeois, and Chris Jermaine. 2021. Tensor Relational Algebra for Distributed Machine Learning System Design. *Proc. VLDB Endow.* 14, 8 (2021), 1338–1350. http://www.vldb.org/pvldb/vol14/p1338-yuan.pdf

[71] Mikhail Yurochkin and XuanLong Nguyen. 2016. Geometric Dirichlet Means Algorithm for topic inference. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 2505–2513. https://proceedings.neurips.cc/paper/2016/hash/a0872cc5b5ca4cc25076f3d868e1bdf8-Abstract.html