Variational Inference for De Finetti Logic

SAMI HADOUAJ, University of Michigan-Dearborn, U.S.A.
OUAEL BEN AMARA, University of Michigan-Dearborn, U.S.A.
NICCOLÒ MENEGHETTI, University of Michigan-Dearborn, U.S.A.

We introduce De Finetti Logic, a novel formalization of the data model of Gamma Probabilistic Databases. De Finetti Logic is a simple, database-centric probabilistic programming framework in which models are specified through relational constraints applied to a probabilistic database. This framework is grounded in the concept of Pólya urns and offers an intuitive formalization of *exchangeability*, the fundamental structural assumption underlying Gamma Probabilistic Databases. We leverage this new formalism to develop a novel inference mechanism based on variational methods. Starting from a DFL theory, our inference method automatically identifies an appropriate family of variational distributions to approximate the target posterior, and synthesizes an optimization algorithm to minimize the Kullback-Leibler divergence between the true posterior and its variational surrogate. To maximize performance, we employ knowledge compilation techniques to limit the number of variational parameters, while preserving the expressive power of the surrogate. Our method is implemented as an extension to the StarfishDB system. Experimental results demonstrate that (i) our approach provides a practical, scalable, and fast-converging alternative to general-purpose inference via collapsed Gibbs sampling, and (ii) remains competitive with specialized algorithms that leverage model-specific optimizations.

CCS Concepts: • **Computing methodologies** → **Probabilistic reasoning**; **Statistical relational learning**; • **Information systems** → *Uncertainty*; • **Mathematics of computing** → *Bayesian computation.*

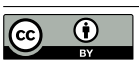
Additional Key Words and Phrases: Probabilistic programming, Datalog, Stochastic variational inference, Posterior inference, Probabilistic databases

ACM Reference Format:
Sami Hadouaj, Ouael Ben Amara, and Niccolò Meneghetti. 2026. Variational Inference for De Finetti Logic. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 237 (June 2026), 30 pages. https://doi.org/10.1145/3802114

1 Introduction

The challenge of performing inference in Bayesian statistical models has been substantially transformed by probabilistic programming (PP) [116]. At its core, PP provides a framework where users specify two components: a stochastic generative process and a set of constraints over the data generated by this process. When the generative distribution is conditioned by the constraints, it yields a posterior probability distribution over the model's latent variables. PP's main advantage is automation. Starting from a generic probabilistic program, a dedicated compiler can automatically generate an inference algorithm to approximate the posterior distribution, eliminating the need for manual development of complex inference procedures. This approach has led to the development of many probabilistic programming languages like Stan [20], Edward [112], Church [53], and Anglican [110], along with various supporting systems and tools [5, 77, 81, 85, 95, 106]. More recently, the integration of first-order logic into probabilistic programming has gained traction in statistical relational learning (SRL) [39, 91–93, 97], enabling structured reasoning over relational

Authors' Contact Information: Sami Hadouaj, shadouaj@umich.edu, University of Michigan-Dearborn, Dearborn, MI, U.S.A.; Ouael Ben Amara, benamara@umich.edu, University of Michigan-Dearborn, Dearborn, MI, U.S.A.; Niccolò Meneghetti, niccolom@umich.edu, University of Michigan-Dearborn, Dearborn, MI, U.S.A..



ACM 2836-6573/2026/6-ART237
https://doi.org/10.1145/3802114

data. Notable contributions to this area include StarfishDB [1] and Probabilistic Programming Datalog (PPDL) [5, 56]. In this line of work, the authors of [1, 84] identified *exchangeability* as a key structural assumption for achieving both expressivity and efficiency. In this paper, we advance this idea by reformulating their framework using the classic Pólya-Eggenberger urn model. This reformulation is intended to clarify and intuitively highlight the central role of exchangeability in probabilistic programs. We refer to this new formalism as De Finetti Logic (DFL). DFL provides a rigorous foundation in which the generative process is modeled as a collection of parallel urns, capturing self-reinforcing dynamics where past events influence future probabilities. Conditioning is implemented by imposing relational constraints over the outcomes of these urns. To prove its effectiveness, we leverage this new formalism to develop a novel stochastic variational inference method. The original implementation of StarfishDB relies on collapsed Gibbs sampling [52] as its primary inference method. This approach inherits the well-known limitations of Markov Chain Monte Carlo (MCMC) techniques [57]: it scales poorly with increasing data size and lacks native support for gradient-based optimization, a cornerstone of modern machine learning. This gap motivates the need for variational inference (VI) [67], a paradigm that replaces intractable posteriors with optimized surrogate distributions [12]. VI's compatibility with stochastic optimization [94] and its ability to handle large datasets make it a natural complement to MCMC in the PP ecosystem. Unfortunately, existing PP frameworks often provide limited support for automating VI in structured relational models. While existing VI frameworks broadly replace intractable posteriors with optimized surrogate distributions [12], they offer limited guidance on tailoring these surrogates to domain-specific languages. This is particularly acute for languages blending logic and probability, where the structure of constraints directly impacts inference efficiency. Existing VI approaches often rely on ad hoc factorizations or exponentially sized parameterizations, limiting their applicability to programs with structured dependencies. We address this gap with the following contributions:

(1) We design a mechanism to derive a stochastic variational inference (SVI) algorithm from any recursion-free, negation-free StarfishDB probabilistic program, enabling scalable optimization for large datasets.
(2) We leverage knowledge compilation techniques to optimize variational inference by reducing the number of required variational parameters, without impacting the expressiveness of the variational family. Furthermore, we employ a restricted version of PPDL to obtain a lifted representation of the conditioning constraints.
(3) We implement our SVI algorithm as an extension of the StarfishDB system and demonstrate its effectiveness through experiments on real-world, large-scale datasets. We also compare it with state-of-the-art logic-based probabilistic programming frameworks.

One key insight of our work is to frame VI as a knowledge compilation task. This approach lets us bound the variational family's complexity while maintaining its expressiveness. As a result, we eliminate the need for manually crafted surrogate distributions, a common practical challenge. Unlike prior VI approaches that either fix factorization [12] or ignore logical structure, our method bridges probabilistic programming with database theory and tractable circuit representations. This connection is aligned with emerging neuro-symbolic paradigms [90, 105].

2 De Finetti Logic

Imagine an urn that initially contains $\alpha_r > 0$ red and $\alpha_g > 0$ green balls. After each draw, the selected ball is replaced in the urn by two balls of the same color. Thus, the probability of drawing a red ball at time step $t + 1$ is equal to $(\alpha_r + R_t)/(\alpha_r + R_t + \alpha_g + G_t)$, where R_t and G_t denote the number of red and green balls drawn in the first t extractions. This urn process, known as the Pólya-Eggenberger urn model [40], was introduced to capture self-reinforcing behavior in stochastic

systems (the more frequently a ball of a certain color is observed in the past, the more likely it is to be drawn in the future). Importantly, this process exhibits the property of *exchangeability* [36]: the probability of observing a given sequence of outcomes remains invariant under permutations of the extraction order, even though the individual events are *not* statistically independent of one another. The following formula determines the probability of extracting a certain sequence S containing exactly R_t red balls and G_t green balls, after performing $t = R_t + G_t$ draws from a Pólya-Eggenberger urn, initialized with α_r red and α_g green balls.

$$P_{\text{Pólya}}(S \mid \alpha_r, \alpha_g) = \frac{\alpha_r^{\overline{R}_t} \cdot \alpha_g^{\overline{G}_t}}{(\alpha_r + \alpha_g)^{\overline{t}}} \tag{1}$$

In Equation (1), $\alpha_r^{\overline{R}_t}$ denotes the rising factorial $\prod_{i=0}^{R_t-1}(\alpha_r + i)$.

Example 1. *A simple application of the Pólya-Eggenberger urn model is in predicting the occurrence of accidents [74]. Consider, for example, Ada, an airplane pilot whose attention level is monitored during each flight by a camera-based system. The system classifies her behavior as either "distracted" (represented by a red ball) or "vigilant" (represented by a green ball). The urn model can then be used to forecast Ada's performance on future flights. Importantly, the prediction depends solely on the number of past observations of each behavior, and not on the specific order in which they occurred.*

Let's now imagine a *parallel* Pólya-Eggenberger urn, consisting of a collection of N pairwise independent urns, from which we perform repeated extraction rounds. At each round t, on every urn a single ball is drawn and then replaced with two additional balls of the same color. A *De Finetti Logic* (DFL) theory is defined as a collection of logical constraints over the outcomes of a parallel Pólya-Eggenberger urn. Each constraint must restrict the outcome of a single extraction round, and no two distinct constraints may refer to the same round.

Example 1 (continued). *Suppose Ada and Bob are two copilots, and we wish to construct a DFL theory to predict their susceptibility to accidents based on past performance. To do this, we instantiate a* parallel *Pólya-Eggenberger urn consisting of $N = 2$ urns, one for Ada and one for Bob. We then carry out several extraction rounds, and encode our prior knowledge about Ada and Bob in the form of DFL constraints. Below is a plausible DFL theory for this scenario. For simplicity, we associate each extraction round with a specific day of the week.*

$\phi_1 \stackrel{\text{def}}{=}$ *"On Monday, if Ada is distracted, then Bob is also distracted."*

$\phi_2 \stackrel{\text{def}}{=}$ *"On Tuesday, Ada is vigilant."*

$\phi_3 \stackrel{\text{def}}{=}$ *"On Wednesday, Ada is distracted."*

$\phi_4 \stackrel{\text{def}}{=}$ *"On Thursday, Ada is vigilant."*

$\phi_5 \stackrel{\text{def}}{=}$ *"On Friday, Bob exhibits the same level of vigilance as Ada."*

With a slight abuse of terminology, we say that the five constraints $\phi_1, \ldots, \phi_5$ from Example 1 are *exchangeable*, in the sense that their likelihood remains unchanged under arbitrary permutations of the constraints over the days of the week. Using more rigorous terminology, we should say that these constraints represent a set of *partially exchangeable events* [33].

Example 1 raises a key question: ($\mathbf{Q}^*$) *Can we utilize the knowledge encoded in the constraints $\phi_1, \ldots, \phi_5$ to predict the future performance of Ada and Bob?* The short and reassuring answer is *yes*, although its rigorous justification requires the introduction of a few more technical concepts and notation.

2.1 Pólya dice

The first step is for us to develop a Bayesian characterization of a Pólya-Eggenberger urn. According to De Finetti's representation theorem [31], any countably infinite sequence S of exchangeable random variables can be represented as a mixture over sequences of independent and identically distributed (i.i.d.) random variables. For an exchangeable sequence S of binary outcomes (e.g., red and green balls), this means that the probability $P(S)$ can be expressed as follows:

$$P(S) = \int_{\mathcal{S}_2} P_{\boldsymbol{\theta}}(S) \cdot \boldsymbol{\mu}(\boldsymbol{\theta})\, d\boldsymbol{\theta} \tag{2}$$

In Equation (2), $\boldsymbol{\theta}$ denotes a latent random two-dimensional vector (θ_r, θ_g) that ranges in the one-dimensional probabilistic simplex $\mathcal{S}_2$, i.e., the set of all pairs of positive real numbers that sum up to one, $\boldsymbol{\mu}$ is a probability measure over $\mathcal{S}_2$, and $P_{\boldsymbol{\theta}}$ represents a probability distribution parameterized by $\boldsymbol{\theta}$, from which the samples in S are drawn independently and identically. If S_i and S_j are random variables corresponding to distinct elements in the sequence S, the distribution of S_i is, by construction, independent from the distribution of S_j when we observe $\boldsymbol{\theta}$ (that is, $S_i \perp S_j \mid \boldsymbol{\theta}$); however, if $\boldsymbol{\theta}$ remains unobserved (i.e., latent), S_i and S_j are only exchangeable, but not necessarily independent (i.e. $S_i \perp S_j$ is not necessarily true). The conditional independence $S_i \perp S_j \mid \boldsymbol{\theta}$ is key: if as we observe more elements of S our uncertainty about $\boldsymbol{\theta}$ decreases, then we can leverage our knowledge about $\boldsymbol{\theta}$ to compute the posterior predictive distribution of the elements of S that we haven't observed yet.

If we use Equation (2) to represent an infinite sequence of draws from a Pólya-Eggenberger urn initialized with α_r red balls and α_g green balls, then the mixing distribution $\boldsymbol{\mu}$ takes the form of a two-dimensional Dirichlet density [78], with concentration parameters (α_r, α_g), and $P_{\boldsymbol{\theta}}(S)$ takes the form of a Bernoulli distribution, that assigns the probability θ_r to the outcome of observing a red ball, and the probability $\theta_g = (1 - \theta_r)$ to the outcome of observing a green ball [40]. This representation remains valid for finite sequences of draws and can be naturally extended to urns with more than two colors [10].

From this point forward, we use the term *Pólya die* to refer to the Bayesian formulation of the Pólya-Eggenberger urn. For any Pólya die $\mathcal{D}_n$, we denote by $\boldsymbol{V}_n = \{v_1, v_2, \ldots, v_{D_n}\}$ the set of possible outcomes from a draw, corresponding to the D_n distinct colors present in the urn. Since DFL establishes a one-to-one correspondence between urn draws and logical constraints, we index the urn draws w.r.t. the constraints they refer to: for each Pólya die $\mathcal{D}_n$, we denote by $\boldsymbol{X}_n = (X_n[\phi_1], X_n[\phi_2], \ldots, X_n[\phi_M])$ an M-vector of random variables modeling the draws associated with constraints $\phi_1, \phi_2, \ldots, \phi_M$. Each random variable in $\boldsymbol{X}_n$ takes values in $\boldsymbol{V}_n$ and represents an independent and identically distributed sample from a shared Categorical probability distribution. The random vector $\boldsymbol{\theta}_n = (\theta_{n,1}, \theta_{n,2}, \ldots, \theta_{n,D_n})$ encodes the weights of this latent Categorical distribution; it lies within the $(D_n{-}1)$-dimensional probabilistic simplex and follows a Dirichlet distribution parameterized by $\boldsymbol{\alpha}_n = (\alpha_{n,1}, \alpha_{n,2}, \ldots, \alpha_{n,D_n})$. Formally, a Pólya die $\mathcal{D}_n$ is defined as the tuple $(\boldsymbol{V}_n, \boldsymbol{X}_n, \boldsymbol{\alpha}_n, \boldsymbol{\theta}_n)$ and obeys the following distributional assumptions:

$$\begin{aligned} \boldsymbol{\theta}_n &\sim \mathcal{D}ir(\boldsymbol{\alpha}_n) \\ X_n[\phi_m] &\sim Cat(\boldsymbol{\theta}_n) \quad \forall X[\phi_m] \in \boldsymbol{X}_n \end{aligned}$$

Thus, a Pólya die defines a simple two-stage probabilistic model. In the first stage, a probability vector $\boldsymbol{\theta}_n$ is sampled from a Dirichlet distribution. In the second stage, $\boldsymbol{\theta}_n$ parameterizes a Categorical distribution from which M samples are drawn. The probability distributions for each stage are

specified as follows:

$$p(\boldsymbol{\theta}_n \mid \boldsymbol{\alpha}_n) = \frac{1}{K(\boldsymbol{\alpha}_n)} \cdot \prod_{v \in \boldsymbol{V}_n} \theta_{n,v}^{\alpha_{n,v}-1} \tag{3}$$

$$P(X_n[\phi_m] \mid \boldsymbol{\theta}_n) = \theta_{n,X_n[\phi_m]} \quad \forall X_n[\phi_m] \in \boldsymbol{X}_n \tag{4}$$

With limited abuse of notation, in Equation (4) we use $X_n[\phi_m]$ to denote either a random variable (as in the first instance of the symbol) or the value assigned to it (as in the second instance). Later, we will adopt the same simplified notation for other random variables, such as $\boldsymbol{X}_n$. In Equation (3) symbol $K(\boldsymbol{\alpha}_n)$ represents the partition function for the Dirichlet distribution, which acts as a normalizing constant:

$$K(\boldsymbol{\alpha}_n) \stackrel{\text{def}}{=} \int_{\mathcal{S}_{D_n}} \prod_{v' \in \boldsymbol{V}_n} \zeta_{v'}^{\alpha_{n,v'}-1} \, d\boldsymbol{\zeta} = \frac{\prod_{v'' \in \boldsymbol{V}_n} \Gamma(\alpha_{n,v''})}{\Gamma\left(\sum_{v' \in \boldsymbol{V}_n} \alpha_{n,v'}\right)} \tag{5}$$

In Equation (5) symbol $\mathcal{S}_{D_n}$ denotes the (D_n-1)-dimensional probabilistic simplex, thus $\boldsymbol{\zeta}$ represents any D_n-vector of positive probabilities that sum up to one. Notice that $\mathcal{S}_{D_n}$ is also the domain of random vector $\boldsymbol{\theta}_n$. $\Gamma(\alpha_{n,v})$, instead, denotes the *Gamma function* $\int_0^{+\infty} z^{\alpha_{n,v}-1} \exp[-z] \; dz$. By combining Equation (3) and Equation (4), we can derive a joint probability distribution over the random variables in $\boldsymbol{X}_n \cup \{\boldsymbol{\theta}_n\}$:

$$p(\boldsymbol{\theta}_n, \boldsymbol{X}_n \mid \boldsymbol{\alpha}_n) = \frac{1}{K(\boldsymbol{\alpha}_n)} \cdot \prod_{v \in \boldsymbol{V}_n} \theta_{n,v}^{\alpha_{n,v}-1} \cdot \prod_{X_n[\phi_m] \in \boldsymbol{X}_n} \theta_{X_n[\phi_m]} \tag{6}$$

It is easy to see that $\boldsymbol{\alpha}_n$ and $\boldsymbol{X}_n$ are independent whenever $\boldsymbol{\theta}_n$ is known (to express this we will write $\boldsymbol{\alpha}_n \perp \boldsymbol{X}_n \mid \boldsymbol{\theta}_n$). If we interpret $\boldsymbol{X}_n$ as a sequence of observed events, then we denote by $C_v(\boldsymbol{X}_n)$ the number of variables in $\boldsymbol{X}_n$ that assume the value v, and by $\boldsymbol{C}(\boldsymbol{X}_n)$ the vector of all counts for each value in $\boldsymbol{V}_n$:

$$\boldsymbol{C}(\boldsymbol{X}_n) \stackrel{\text{def}}{=} (C_{v_1}(\boldsymbol{X}_n), C_{v_2}(\boldsymbol{X}_n), \ldots, C_{v_{D_n}}(\boldsymbol{X}_n))$$

Under this interpretation, it is easy to rewrite Equation (6) as follows:

$$p(\boldsymbol{\theta}_n, \boldsymbol{X}_n \mid \boldsymbol{\alpha}_n) = p(\boldsymbol{\theta}_n \mid \boldsymbol{\alpha}_n) \cdot \prod_{v \in \boldsymbol{V}_n} \theta_{n,v}^{C_v(\boldsymbol{X}_n)} \tag{7}$$

From Equation (7) we can conclude that the random variables in $\boldsymbol{X}_n$ are pairwise *finitely exchangeable* [32], i.e. their probability distribution remains invariant w.r.t. permutations, as we expected. Since the Dirichlet distribution acts as a conjugate prior to the Categorical distribution, the *posterior density* of a Pólya die is another Dirichlet distribution:

$$p(\boldsymbol{\theta}_n \mid \boldsymbol{X}_n, \boldsymbol{\alpha}_n) \sim \mathcal{D}ir(\boldsymbol{\alpha}_n + \boldsymbol{C}(\boldsymbol{X}_n)) \tag{8}$$

The *posterior predictive* of a Pólya die is instead a Categorical distribution. If we denote by $\boldsymbol{X}_n[1..m]$ the vector obtained by considering only the first m elements of vector $\boldsymbol{X}_n$, the *posterior predictive* of a Pólya die can be computed as follows:

$$P(X_n[\phi_{m+1}] \mid \boldsymbol{X}_n[1..m], \boldsymbol{\alpha}_n) = \frac{\alpha_{n,X_n[\phi_{m+1}]} + C_{X_n[\phi_{m+1}]}(\boldsymbol{X}_n[1..m])}{\sum_{v' \in \boldsymbol{V}_n} (\alpha_{n,v'} + C_{v'}(\boldsymbol{X}_n[1..m]))}$$

Finally, we can conclude that the likelihood of observing the sequence of M events $\boldsymbol{X}_n$ out of a Pólya die is equivalent to the likelihood of observing $\boldsymbol{X}_n$ out of M extractions from a Pólya-Eggenberger

urn [75]:

$$\begin{aligned} P(\boldsymbol{X}_n \mid \boldsymbol{\alpha}_n) &= \int_{\mathcal{S}_{D_n}} p(\boldsymbol{\theta}_n, \boldsymbol{X}_n \mid \boldsymbol{\alpha}_n)\, d\boldsymbol{\theta}_n \\ &= P(X_n[\phi_1] \mid \boldsymbol{\alpha}_n) \cdot \prod_{m=1}^{M-1} P(X_n[\phi_{m+1}] \mid \boldsymbol{X}_n[1..m], \boldsymbol{\alpha}_n) \\ &= \frac{\prod_{v \in \boldsymbol{V}_n} \left(\alpha_{n,v}^{\overline{C_v(\boldsymbol{X}_n)}}\right)}{\left(\sum_{v \in \boldsymbol{V}_n} \alpha_{n,v}\right)^{\overline{M}}} \end{aligned} \tag{9}$$

Thus, the random variables in $\boldsymbol{X}_n$ are exchangeable, *but they are not pairwise independent*, since the probability distribution of each variable depends explicitly on the others; this means that observing any subset of variables in $\boldsymbol{X}_n$ will affect our knowledge about the others. We conclude by noticing that the likelihood of a single urn extraction takes the form of a Categorical distribution:

$$P(X_n[\phi_1] \mid \boldsymbol{\alpha}_n) = \int_{\mathcal{S}_{D_n}} p(\boldsymbol{\theta}_n, X_n[\phi_1] \mid \boldsymbol{\alpha}_n)\, d\boldsymbol{\theta}_n = \frac{\alpha_{n,X_n[\phi_1]}}{\sum_{v \in \boldsymbol{V}_n} \alpha_{n,v}}$$

In summary, a Pólya die $\mathcal{D}_n = (\boldsymbol{V}_n, \boldsymbol{X}_n, \boldsymbol{\alpha}_n, \boldsymbol{\theta}_n)$ serves as a basic Bayesian model for a Pólya-Eggenberger urn: $\boldsymbol{V}_n$ specifies the set of possible colors that can be drawn, $\boldsymbol{\alpha}_n$ encodes the initial composition of the urn, and $\boldsymbol{X}_n$ records the sequence of draws and their outcomes. The distribution of the latent vector $\boldsymbol{\theta}_n$ summarizes the information gathered from previous draws and enables us to compute the posterior predictive distribution for future draws. Following common practice, we identify $\boldsymbol{\alpha}_n$ and $\boldsymbol{\theta}_n$ as vectors of *hyperparameters* and *latent parameters*, respectively.

2.2 DFL Theories

The second step to answer question $\mathbf{Q}^*$ is to formalize the language that we use to define constraints over the outcomes of a parallel Pólya-Eggenberger urn, that we model as a battery of N pair-wise independent Pólya dice $\{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$. We denote by $\mathbb{A}$ the set of all the hyperparameters, by Θ the set of all latent parameters, by $\mathbb{X}$ the set of all throws, by $\mathbb{V}$ the set of all domains:

$$\mathbb{A} \stackrel{\text{def}}{=} \{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N\} \qquad \Theta \stackrel{\text{def}}{=} \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}$$
$$\mathbb{X} \stackrel{\text{def}}{=} \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\} \qquad \mathbb{V} \stackrel{\text{def}}{=} \{\boldsymbol{V}_1, \ldots, \boldsymbol{V}_N\}$$

For a given model $\mathcal{G} \stackrel{\text{def}}{=} (\Theta, \mathbb{X}, \mathbb{A})$ we define a joint probability distribution over the random variables in $\mathbb{X}$ and Θ as follows:

$$p(\Theta, \mathbb{X} \mid \mathbb{A}) = \prod_{n=1}^{N} p(\boldsymbol{\theta}_n, \boldsymbol{X}_n \mid \boldsymbol{\alpha}_n) \propto \prod_{n=1}^{N} \left[\prod_{v \in \boldsymbol{V}_n} \theta_{n,v}^{\alpha_{n,v}-1} \cdot \prod_{X_n[\phi_m] \in \boldsymbol{X}_n} \theta_{X_n[\phi_m]} \right]$$

It is easy to see that $\mathcal{G}$ respects the following structural assumption: $\mathbb{A} \perp \mathbb{X} \mid \Theta$. A DFL theory consists of a finite set $\Phi \stackrel{\text{def}}{=} \{\phi_1, \ldots, \phi_M\}$ of logical constraints that are imposed against a probabilistic model $\mathcal{G} = (\Theta, \mathbb{X}, \mathbb{A})$. All the constraints in Φ must be satisfiable, and, as previously noted, each constraint must restrict the outcome of exactly one extraction round, and no two distinct constraints may refer to the same round.

Constraints come in two variants: *atomic* and *composite*. An atomic constraint simply limits the possible outcomes of a specific urn draw, for example:

$$\phi_1 := (X_1[\phi_1] \in \{v_2, v_3\})$$

When an atomic constraint allows only for one value (e.g. $X_1[\phi_1] \in \{v_0\}$) we say that the constraint is *simple* and write $X_1[\phi_1] = v_0$. Furthermore, we use the assignment operator := to associate constraints with their label, as to avoid any confusion with the equality constraint (=). Composite constraints are obtained by combining multiple atomic ones using the logical operators $\wedge$ ("and"), $\vee$ ("or"), $\neg$ ("not").

Example 1 (continued). *We are now ready to reformulate our DFL theory as a well-defined Bayesian model. To do so, we instantiate two Pólya dice, one for Ada and one for Bob, namely* $\mathcal{D}_A \stackrel{\text{def}}{=} (\boldsymbol{V}_A, \boldsymbol{X}_A, \boldsymbol{\alpha}_A, \boldsymbol{\theta}_A)$ *and* $\mathcal{D}_B \stackrel{\text{def}}{=} (\boldsymbol{V}_B, \boldsymbol{X}_B, \boldsymbol{\alpha}_B, \boldsymbol{\theta}_B)$*, with* $\boldsymbol{V}_A = \boldsymbol{V}_B = \{$*distracted*, *vigilant*$\}$*. We can then rewrite our DFL constraints as follows.*

$$\begin{aligned}
\phi_1 &:= (\neg(X_A[\phi_1] = \textit{distracted}) \vee (X_B[\phi_1] = \textit{distracted})) \\
\phi_2 &:= (X_A[\phi_2] = \textit{vigilant}) \\
\phi_3 &:= (X_A[\phi_3] = \textit{distracted}) \\
\phi_4 &:= (X_A[\phi_4] = \textit{vigilant}) \\
\phi_5 &:= ((X_A[\phi_5] = \textit{distracted}) \wedge (X_B[\phi_5] = \textit{distracted})) \vee \\
&\quad \vee ((X_A[\phi_5] = \textit{vigilant}) \wedge (X_B[\phi_5] = \textit{vigilant}))
\end{aligned} \tag{10}$$

If ϕ_m is a constraint, we denote by Θ_{ϕ_m} and $\textsc{Var}(\phi_m)$ all the latent parameters and urn-draws, respectively, of the dice that are mentioned in ϕ_m. For example, if we consider ϕ_1 from Equation (10) in Example 1 then $\Theta_{\phi_1} = \{\boldsymbol{\theta}_A, \boldsymbol{\theta}_B\}$ and $\textsc{Var}(\phi_1) = \{X_A[\phi_1], X_B[\phi_1]\}$. We often write $\textsc{Var}(\phi_m, \phi_{m'})$ as a shorthand for $\textsc{Var}(\phi_m) \cup \textsc{Var}(\phi_{m'})$.

A Boolean constraint is called a *term expression* if it is a conjunction ($\wedge$) of *simple* atomic constraints, where each variable appears in at most one of the simple constraints. We use the symbol τ to denote an arbitrary term expression, and we write $\tau\langle X_n[\phi_m]\rangle$ to indicate the value assigned by τ to the random variable $X_n[\phi_m]$. In Example 1 the constraint ϕ_5 from Equation (10) consists of a disjunction ($\vee$) between two term expressions

$$\begin{aligned}
\tau_1 &= ((X_A[\phi_5] = \text{distracted}) \wedge (X_B[\phi_5] = \text{distracted})) \\
\tau_2 &= ((X_A[\phi_5] = \text{vigilant}) \wedge (X_B[\phi_5] = \text{vigilant}))
\end{aligned}$$

Thus $\tau_1\langle X_A[\phi_5]\rangle$ evaluates to *distracted*, while $\tau_2\langle X_A[\phi_5]\rangle$ evaluates to *vigilant*.

Given an arbitrary subset $\mathbb{X}' \subseteq \mathbb{X}$, we denote by $\textsc{Asst}(\mathbb{X}')$ the set of all possible configurations (assignments) of the variables in $\mathbb{X}'$. For convenience, we often represent these assignments as term expressions. Adopting the terminology of [26], we refer to $\textsc{Asst}(\mathbb{X})$ as the set of all the *possible worlds* of process $\mathcal{G}$. If $\mathbb{X}'$ satisfies $\textsc{Var}(\phi) \subseteq \mathbb{X}' \subseteq \mathbb{X}$, we denote by $\textsc{Sat}(\phi, \mathbb{X}')$ the subset of $\textsc{Asst}(\mathbb{X}')$ where constraint ϕ is satisfied. We say that two constraints ϕ_1 and ϕ_2 are logically equivalent, and write $\phi_1 \equiv \phi_2$, when they are satisfied by exactly the same assignments, i.e. $\textsc{Sat}(\phi_1, \textsc{Var}(\phi_1, \phi_2)) = \textsc{Sat}(\phi_2, \textsc{Var}(\phi_1, \phi_2))$. We say that constraint ϕ_1 *entails* constraint ϕ_2 (and write $\phi_1 \models \phi_2$) when $\textsc{Sat}(\phi_1, \textsc{Var}(\phi_1, \phi_2)) \subseteq \textsc{Sat}(\phi_2, \textsc{Var}(\phi_1, \phi_2))$.

If we consider constraint ϕ_5 from Equation (10) in Example 1 and define τ_3 and τ_4 as follows

$$\begin{aligned}
\tau_3 &= ((X_A[\phi_5] = \text{distracted}) \wedge (X_B[\phi_5] = \text{vigilant})) \\
\tau_4 &= ((X_A[\phi_5] = \text{vigilant}) \wedge (X_B[\phi_5] = \text{distracted}))
\end{aligned}$$

then $\textsc{Asst}(\textsc{Var}(\phi_5))$ and $\textsc{Sat}(\phi_5, \textsc{Var}(\phi_5))$ denote the sets $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ and $\{\tau_1, \tau_2\}$, respectively. Furthermore, $\tau_1 \models \phi_5$ and $\phi_5 \equiv \neg\tau_3 \wedge \neg\tau_4$. If Φ is a DFL theory, we define $\textsc{Sat}(\Phi, \mathbb{X})$ as follows:

$$\textsc{Sat}(\Phi, \mathbb{X}) \stackrel{\text{def}}{=} \{\tau_1 \wedge \ldots \wedge \tau_M \mid (\tau_1, \ldots, \tau_M) \in \times_{\phi\in\Phi}\textsc{Sat}(\phi, \textsc{Var}(\phi))\}$$

Intuitively, the elements of $\textsc{Sat}(\Phi, \mathbb{X})$ are term expressions of the form $\tau_1 \wedge \ldots \wedge \tau_M$, each representing a sequence of M outcomes $(\tau_1, \ldots, \tau_M)$ obtained by performing M draws from a set of N pairwise independent Pólya-Eggenberger urns, where each outcome τ_m in $(\tau_1, \ldots, \tau_M)$ satisfies the corresponding constraint ϕ_m in Φ.

Equivalently, $\textsc{Sat}(\Phi, \mathbb{X})$ denotes the subset of possible worlds in $\textsc{Asst}(\mathbb{X})$ where all constraints in Φ are satisfied. With a slight abuse of notation, we will use the symbol Φ to refer both to a DFL theory (i.e., a set of DFL constraints) and to the expression $\bigvee_{\tau \in \textsc{Sat}(\Phi, \mathbb{X})} \tau$. Note that Φ (when interpreted as an expression) and the term expressions in $\textsc{Asst}(\mathbb{X})$ and $\textsc{Sat}(\Phi, \mathbb{X})$ are not valid DFL constraints, as they refer to M distinct draws rather than a single draw. Nonetheless, their likelihood remains well-defined and easy to compute, through Equation (9).

It is easy to see that any satisfiable constraint ϕ with $\textsc{Var}(\phi) \subseteq \mathbb{X}$ encodes an *event* in the probability space defined by $\mathcal{G}$. We denote by $P(\phi \mid \mathbb{A})$ the probability of sampling from $\mathcal{G}$ a possible world that satisfies constraint ϕ

$$P(\phi \mid \mathbb{A}) \stackrel{\text{def}}{=} \sum_{\tau \in \textsc{Sat}(\phi, \mathbb{X})} P(\tau \mid \mathbb{A}) \tag{11}$$

Furthermore, any event of $\mathcal{G}$ may be conditioned w.r.t. a constraint. We will denote by $P(\phi_1 \mid \phi_2, \mathbb{A})$ the probability of sampling a possible world that satisfies constraint ϕ_1 from the set of possible worlds that satisfy ϕ_2.

$$P(\phi_1 \mid \phi_2, \mathbb{A}) \stackrel{\text{def}}{=} \frac{P(\phi_1 \wedge \phi_2 \mid \mathbb{A})}{P(\phi_2 \mid \mathbb{A})}$$

We say that two constraints ϕ_1 and ϕ_2 are *independent* if their joint distribution is equal to the product of $P(\phi_1|\mathbb{A}) \cdot P(\phi_2|\mathbb{A})$. Two constraints ϕ_1 and ϕ_2 are partially exchangeable [37] when $P(\phi_1, \phi_2 \mid \Theta_{\phi_1}, \Theta_{\phi_2}) = P(\phi_1 \mid \Theta_{\phi_1}) \cdot P(\phi_2 \mid \Theta_{\phi_2})$. For simplicity, and with limited abuse of terminology, in this paper we will often omit the adverb "partially" and claim that two constraints are simply exchangeable when they belong to a partially exchangeable set. Notice that $\Theta_{\phi_1} \cap \Theta_{\phi_2} = \emptyset$ is a sufficient condition for ϕ_1 and ϕ_2 to be independent and $\textsc{Var}(\phi_1) \cap \textsc{Var}(\phi_2) = \emptyset$ is a sufficient condition for ϕ_1 and ϕ_2 to be partially exchangeable. Thus, all the constraints in a DFL theory are always, by definition, pairwise partially exchangeable.

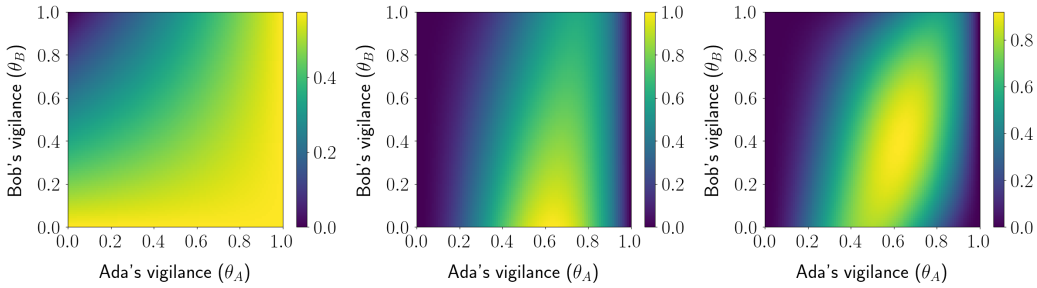
2.3 Posterior Inference

We are now ready to answer question $\mathbf{Q}^*$. Let's denote by $\mathcal{P} = (\mathcal{G}, \Phi)$ an arbitrary DFL program, where $\mathcal{G} = (\Theta, \mathbb{X}, \mathbb{A})$ is a generative model and Φ is a set of exchangeable constraints over $\mathcal{G}$. Our goal is to leverage the partial exchangeability of the constraints in Φ to compute the posterior distribution of the latent parameters in Θ, conditioned on the constraints Φ and the model's parameters $\mathbb{A}$.

Theorem 1. *[1] For any well-formed DFL program $\mathcal{P} = (\mathcal{G}, \Phi)$ where $\mathcal{G} = (\Theta, \mathbb{X}, \mathbb{A})$, the posterior density $p(\Theta \mid \Phi, \mathbb{A})$ can be computed as an affine combination of products of Dirichlet densities.*

$$p(\Theta \mid \Phi, \mathbb{A}) = \sum_{\tau \in \mathit{SAT}(\Phi, \mathbb{X})} p(\Theta \mid \tau, \mathbb{A}) \cdot P(\tau \mid \Phi, \mathbb{A}) \tag{12}$$

Equation (12) admits an intuitive interpretation: each term τ in the sum corresponds to a plausible sequence of M draws from an N-ary parallel Pólya-Eggenberger urn that satisfies all constraints in Φ. The probability $P(\tau \mid \Phi, \mathbb{A})$ denotes the likelihood of observing such an outcome among all sequences that satisfy Φ, while $p(\Theta \mid \tau, \mathbb{A})$ represents the posterior distribution of Θ conditioned on that outcome. Owing to the conjugacy of the Dirichlet prior, this posterior density is given by a product of Dirichlet distributions, as specified in Equation (8): $p(\Theta \mid \tau, \mathbb{A}) = \prod_{\boldsymbol{\theta}_n \in \Theta} p(\boldsymbol{\theta}_n \mid \tau, \boldsymbol{\alpha}_n)$.



(a) $p(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B \mid \{\phi_1\}, \boldsymbol{\alpha}_A, \boldsymbol{\alpha}_B)$ **(b)** $p(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B | \{\phi_1, \ldots, \phi_4\}, \boldsymbol{\alpha}_A, \boldsymbol{\alpha}_B)$ **(c)** $p(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B | \{\phi_1, \ldots, \phi_5\}, \boldsymbol{\alpha}_A, \boldsymbol{\alpha}_B)$

Fig. 1. Posterior distributions induced by the probabilistic program from Example 1

It is immediate to derive from Equation (12) the full posterior $p(\Theta, \mathbb{X} = \tau | \Phi, \mathbb{A}) = p(\Theta | \tau, \mathbb{A}) \cdot P(\tau | \Phi, \mathbb{A})\ \forall \tau \in \textsc{Sat}(\Phi, \mathbb{X})$. The proof of Equation (12) is provided in the supplemental appendix.

Example 1 (continued). *Figure 1 illustrates the posterior distributions generated by three subsets of constraints from the DFL theory outlined in (10). Each point (θ_A, θ_B) in the plots represents two probabilities, the likelihood of vigilance of Ada (θ_A) and Bob (θ_B). The color gradient indicates the posterior density of each (θ_A, θ_B) pair: areas shaded in blue correspond to low posterior density, while areas shaded in yellow correspond to high posterior density. Figure 1a shows the posterior distribution generated by the first constraint alone, ϕ_1, which suggests that Ada's distraction may propagate to Bob. Note that the top-left quadrant, which contains configurations likely to violate this constraint, is shaded in blue. Figure 1b shows the posterior distribution generated by the first four constraints in the theory (10), where constraints $\{\phi_2, \phi_3, \phi_4\}$ state that Ada is vigilant about two out of three times. Note that this additional information is reflected in the posterior distribution, where the centroid of the marginal distribution of θ_A is shifted towards the value $\frac{2}{3}$. Moreover, Bob results to be more likely to be distracted than vigilant, which is consistent with constraint ϕ_1. Figure 1c shows the posterior distribution generated by the whole DFL theory (10). Note that Bob's likelihood of vigilance is improved, consistently with constraint ϕ_5. We conclude this example by noting that the logical constraints in the DFL theory induce correlations between the posterior densities of θ_A and θ_B. These correlations arise from the logical interdependencies among the constraints defined in the theory.*

The primary inference task in a DFL program is to compute the posterior distributions of Θ and $\mathbb{X}$. Unfortunately, Equation (12) is of limited practical use in this task, as the set $\textsc{Sat}(\Phi, \mathbb{X})$ may grow exponentially with the size of Φ. The authors of [84] addressed this limitation by introducing a Markov Chain Monte Carlo method, able to approximate $p(\Theta \mid \Phi, \mathbb{A})$ for very large probabilistic programs, that may contain millions of constraints. Furthermore, the authors of [1, 84] show that the language is expressive enough to encode non-trivial Bayesian models, such as the Ising model, Latent Dirichlet Allocation [13] or Hidden Markov Models. In the next section we introduce a novel, alternative inference technique, based on Variational Inference [12]. Compared to standard MCMC methods, Variational Inference offers better scalability and guaranteed convergence, though at the expense of accuracy.

Before proceeding, we want to emphasize the significance and impact of the problem at hand. Thanks to its ability to encode arbitrary propositional logic constraints into a smooth posterior distribution, DFL provides a new, alternative way to combine logical and statistical AI into a single

unified framework, similarly to Markov Logic Networks [38], ProbLog [68] or Probabilistic Soft Logic [4].

3 Mean Field Variational Inference

In this section, we introduce a simple variational method to approximate the joint posterior distribution $p(\Theta, \mathbb{X} \mid \Phi, \mathbb{A})$ for any arbitrary probabilistic program $(\mathcal{G}, \Phi)$. This method consists of two steps: first we design a variational distribution $q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M})$ that has the same support as the posterior $p(\Theta, \mathbb{X} \mid \Phi, \mathbb{A})$ but a different parametrization and simplified structural assumptions; second we optimize the parameters of the variational distribution $q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M})$ with the goal of minimizing its KL-divergence w.r.t. $p(\Theta, \mathbb{X} \mid \Phi, \mathbb{A})$. The variational distribution $q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M})$ is defined as follows:

(1) For each constraint $\phi_m \in \Phi$ we define a variational Categorical distribution $q_{\phi_m}(\textsc{Var}(\phi_m) \mid \boldsymbol{\lambda}_m)$ that assigns a probability to each term in $\textsc{Asst}(\textsc{Var}(\phi_m))$. Importantly, q_{ϕ_m} is a full joint distribution over $\textsc{Var}(\phi_m)$ that does not make any independence assumption about the random variables in $\textsc{Var}(\phi_m)$. We denote by Λ the set $\cup_{\phi_m \in \Phi}\{\boldsymbol{\lambda}_m\}$ containing all the Categorical variational parameters. We refer to these parameters as *local* parameters.
(2) For each die $\mathcal{D}_n \in \mathcal{G}$ we define a variational distribution $q_{\mathcal{D}_n}(\boldsymbol{\theta}_n \mid \boldsymbol{\mu}_n)$, a Dirichlet density over the domain of random vector $\boldsymbol{\theta}_n$. We denote by $\mathbb{M}$ the set $\cup_{\mathcal{D}_n \in \mathcal{G}}\{\boldsymbol{\mu}_n\}$, and refer to these parameters as *global* parameters.
(3) We define the variational posterior $q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M})$ as

$$q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M}) \stackrel{\text{def}}{=} q_\Phi(\mathbb{X} \mid \Lambda) \cdot q_{\mathcal{G}}(\Theta \mid \mathbb{M}) \text{ where}$$
$$q_\Phi(\mathbb{X} \mid \Lambda) \stackrel{\text{def}}{=} \prod_{\phi_m \in \Phi} q_{\phi_m}(\textsc{Var}(\phi_m) \mid \boldsymbol{\lambda}_m),$$
$$q_{\mathcal{G}}(\Theta \mid \mathbb{M}) \stackrel{\text{def}}{=} \prod_{\mathcal{D}_n \in \mathcal{G}} q_{\mathcal{D}_n}(\boldsymbol{\theta}_n \mid \boldsymbol{\mu}_n)$$

Our goal is to identify the values of Λ and $\mathbb{M}$ that minimize the KL divergence between $q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M})$ and the joint posterior

$$p(\Theta, \mathbb{X} \mid \Phi, \mathbb{A}) = p(\Theta \mid \mathbb{X}, \mathbb{A}) \cdot P(\mathbb{X} \mid \Phi, \mathbb{A})$$

The KL divergence is defined as follows

$$\begin{aligned} KL(q, p) &\stackrel{\text{def}}{=} \langle \log \frac{q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M})}{p(\Theta, \mathbb{X} \mid \Phi, \mathbb{A})} \rangle_{(\Theta, \mathbb{X}) \sim q} \\ &= \langle \log q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M}) \rangle_{(\Theta, \mathbb{X}) \sim q} - \langle \log p(\Theta, \mathbb{X} \mid \Phi, \mathbb{A}) \rangle_{(\Theta, \mathbb{X}) \sim q} \\ &= \langle \log q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M}) \rangle_{(\Theta, \mathbb{X}) \sim q} - \langle \log \frac{p(\Theta, \mathbb{X}, \Phi \mid \mathbb{A})}{p(\Phi \mid \mathbb{A})} \rangle_{(\Theta, \mathbb{X}) \sim q} \\ &= \langle \log q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M}) \rangle_{(\Theta, \mathbb{X}) \sim q} - \langle \log p(\Theta, \mathbb{X}, \Phi \mid \mathbb{A}) \rangle_{(\Theta, \mathbb{X}) \sim q} + \langle \log p(\Phi \mid \mathbb{A}) \rangle_{(\Theta, \mathbb{X}) \sim q} \end{aligned}$$

Here we denote by $\langle f(z) \rangle_{z\ p(z)}$ the expected value of function $f(z)$ assuming that z is distributed as per $p(z)$. As per common practice [12], instead of minimizing the KL divergence directly, a problem that is intractable, we will settle instead on maximizing the *evidence lower bound* ($\mathcal{L}$) between $q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M})$ and $p(\Theta, \mathbb{X} \mid \Phi, \mathbb{A})$.

$$\mathcal{L}[q(\Lambda, \mathbb{M})] \stackrel{\text{def}}{=} \langle \log p(\Theta, \mathbb{X}, \Phi \mid \mathbb{A}) \rangle_{(\Theta, \mathbb{X}) \sim q} - \langle \log q(\Theta, \mathbb{X} \mid \Lambda, \mathbb{M}) \rangle_{(\Theta, \mathbb{X}) \sim q}$$

The term $\log p(\Theta, \mathbb{X}, \Phi \mid \mathbb{A})$ can be decomposed as follows

$$\begin{aligned}\log p(\Theta, \mathbb{X}, \Phi \mid \mathbb{A}) &= \log\left[p(\Theta \mid \mathbb{A}) \cdot p(\mathbb{X} \mid \Theta) \cdot p(\Phi \mid \mathbb{X})\right] \\ &= \log p(\Theta \mid \mathbb{A}) + \log p(\mathbb{X} \mid \Theta) + \log p(\Phi \mid \mathbb{X})\end{aligned}$$

Therefore, we can rewrite the ELBO as follows

$$\begin{aligned}\mathcal{L}[q(\Lambda, \mathbb{M})] = \langle \log p(\Theta \mid \mathbb{A})\rangle_{\Theta\sim q_{\mathcal{G}}} + \\ + \langle \log p(\mathbb{X} \mid \Theta)\rangle_{(\Theta,\mathbb{X})\sim q} + \langle \log p(\Phi \mid \mathbb{X})\rangle_{\mathbb{X}\sim q_\Phi} + \\ - \langle \log q_\Phi(\mathbb{X} \mid \Lambda)\rangle_{\mathbb{X}\sim q_\Phi} - \langle \log q_{\mathcal{G}}(\Theta \mid \mathbb{M})\rangle_{\Theta\sim q_{\mathcal{G}}}\end{aligned}$$

Notice that the term $\langle \log p(\Phi \mid \mathbb{X})\rangle_{\mathbb{X}\sim q_\Phi}$ diverges to $-\infty$ if we allow distribution q_Φ to generate with positive probability any possible world that does not satisfy Φ. Thus, to maximize $\mathcal{L}[q(\Lambda, \mathbb{M})]$ we need to restrict Λ so that any assignment that does not satisfy Φ has probability 0. We denote by $q'_\Phi(\mathbb{X} \mid \Lambda')$ this new, restricted variational distribution

$$\begin{aligned}\mathcal{L}[q'(\Lambda', \mathbb{M})] = \langle \log p(\Theta \mid \mathbb{A})\rangle_{\Theta\sim q_{\mathcal{G}}} + \langle\langle \log p(\mathbb{X} \mid \Theta)\rangle_{\Theta\sim q_{\mathcal{G}}}\rangle_{\mathbb{X}\sim q'_\Phi} + \\ - \langle \log q'_\Phi(\mathbb{X} \mid \Lambda')\rangle_{\mathbb{X}\sim q'_\Phi} - \langle \log q_{\mathcal{G}}(\Theta \mid \mathbb{M})\rangle_{\Theta\sim q_{\mathcal{G}}}\end{aligned}$$

If we express $\mathcal{L}[q'(\Lambda', \mathbb{M})]$ as a function of a single variational parameter $\boldsymbol{\lambda}'_m$, then we have

$$\mathcal{L}[q'_{\phi_m}(\boldsymbol{\lambda}'_m)] = \langle\langle \log p(\tau_m \mid \Theta)\rangle_{\Theta\sim q_{\mathcal{G}}}\rangle_{\tau_m\sim q'_{\phi_m}} - \langle \log q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m)\rangle_{\tau_m\sim q'_{\phi_m}} + \text{const.} \tag{13}$$

Here τ_m denotes any term expression from $\textsc{Sat}(\phi_m, \textsc{Var}(\phi_m))$. We compute the functional derivative of $\mathcal{L}[q'_{\phi_m}(\boldsymbol{\lambda}'_m)]$ w.r.t. q'_{ϕ_m}

$$\frac{\partial \mathcal{L}[q'_{\phi_m}(\boldsymbol{\lambda}'_m)]}{\partial q'_{\phi_m}} = \langle \log p(\tau_m \mid \Theta)\rangle_{\Theta\sim q_{\mathcal{G}}} - \log q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m) - 1 \tag{14}$$

The proof of Equation (14) is provided in the supplemental appendix. By setting the functional derivative to zero we obtain

$$\begin{aligned}\log q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m) &\propto \langle \log p(\tau_m \mid \Theta)\rangle_{\Theta\sim q_{\mathcal{G}}} \\ &\propto \sum_{X_n[\phi_m]\in\textsc{Var}(\phi_m)} \langle \log p(\tau_m\langle X_n[\phi_m]\rangle \mid \boldsymbol{\theta}_n)\rangle_{\boldsymbol{\theta}_n\sim q_{\mathcal{D}_n}} \\ &\propto \sum_{X_n[\phi_m]\in\textsc{Var}(\phi_m)} (\psi(\mu_{n,\tau_m\langle X_n[\phi_m]\rangle}) - \sum_{v\in\boldsymbol{V}_n} \psi(\mu_{n,v}))\end{aligned}$$

Thus we can optimize λ'_{m,τ_m} as follows

$$\lambda'_{m,\tau_m} \propto \exp \sum_{X_n[\phi_m]\in\textsc{Var}(\phi_m)} \left[\psi(\mu_{n,\tau_m\langle X_n[\phi_m]\rangle}) - \sum_{v\in\boldsymbol{V}_n} \psi(\mu_{n,v})\right] \tag{15}$$

We can apply a similar derivation to optimize the variational parameters in $\mathbb{M}$. If we express the $\mathcal{L}[q_{\mathcal{D}_n}(\boldsymbol{\mu}_n)]$ as function of a single variational parameter $\boldsymbol{\mu}_n$ then we have:

$$\begin{aligned}\mathcal{L}[q_{\mathcal{D}_n}(\boldsymbol{\mu}_n)] = \langle \log p(\boldsymbol{\theta}_n \mid \boldsymbol{\alpha}_n)\rangle_{\boldsymbol{\theta}_n\sim q_{\mathcal{D}_n}} + \\ + \sum_{\phi_m\in\Phi} \sum_{X_n[\phi_m]\in\textsc{Var}(\phi_m)} \langle\langle \log p(\boldsymbol{X}_n \mid \boldsymbol{\theta}_n)\rangle_{\mathbb{X}\sim q'_\Phi}\rangle_{\boldsymbol{\theta}_n\sim q_{\mathcal{D}_n}} + \\ - \langle \log q(\boldsymbol{\theta}_n \mid \boldsymbol{\mu}_n)\rangle_{\boldsymbol{\theta}_n\sim q_{\mathcal{D}_n}} + \text{const.}\end{aligned} \tag{16}$$

We compute the functional derivative of $\mathcal{L}[q_{\mathcal{D}_n}(\boldsymbol{\mu}_n)]$ w.r.t $q_{\mathcal{D}_n}$

$$\frac{\partial \mathcal{L}[q_{\mathcal{D}_n}(\boldsymbol{\mu}_n)]}{\partial q_{\mathcal{D}_n}} = \log p(\boldsymbol{\theta}_n \mid \boldsymbol{\alpha}_n) + \langle \log p(\boldsymbol{X}_n \mid \boldsymbol{\theta}_n) \rangle_{\mathbb{X}\sim q'_\Phi} - \log q(\boldsymbol{\theta}_n \mid \boldsymbol{\mu}_n) - 1 \tag{17}$$

By setting the functional derivative to zero we obtain

$$\begin{aligned}\log q(\boldsymbol{\theta}_n \mid \boldsymbol{\mu}_n) &= \log p(\boldsymbol{\theta}_n \mid \boldsymbol{\alpha}_n) + \langle \log p(\boldsymbol{X}_n \mid \boldsymbol{\theta}_n) \rangle_{\mathbb{X}\sim q'_\Phi} - 1 \\ &= \log(K(\boldsymbol{\alpha}_n)) - 1 + \sum_{v \in \boldsymbol{V}_n} (\alpha_{n,v} - 1) \cdot \log(\theta_{n,v}) + \\ &\quad + \sum_{v \in \boldsymbol{V}_n} \sum_{\phi_m \in \Phi} \sum_{\tau_m \in \textsc{Sat}(\phi_m)} \log(\theta_{n,v}) \cdot C_v(\tau_m \langle X_n[\phi_m] \rangle) \cdot q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m)\end{aligned}$$

Here $K(\boldsymbol{\alpha}_n)$ is the normalization constant of the Dirichlet distribution parametrized by the vector $\boldsymbol{\alpha}_n$.

$$\log q(\boldsymbol{\theta}_n \mid \boldsymbol{\mu}_n) \propto \sum_{v \in \boldsymbol{V}_n} \log(\theta_{n,v}) \Big[\sum_{\phi_m \in \Phi} \sum_{\tau_m \in \textsc{Sat}(\phi_m)} C_v(\tau_m \langle X_n[\phi_m] \rangle) \cdot q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m) + (\alpha_{n,v} - 1) \Big]$$

By definition, $q(\boldsymbol{\theta}_n \mid \boldsymbol{\mu}_n)$ is a Dirichlet distribution, hence we have

$$\sum_{v \in \boldsymbol{V}_n} \log \theta_{n,v}^{\mu_{n,v}-1} \propto \sum_{v \in \boldsymbol{V}_n} \log \theta_{n,v} \cdot \left(\sum_{\phi_m \in \Phi} \sum_{\tau_m \in \textsc{Sat}(\phi_m)} C_v(\tau_m \langle X_n[\phi_m] \rangle) \cdot q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m) + \alpha_{n,v} - 1 \right)$$

Thus we can optimize $\mu_{n,v}$ as follows

$$\mu_{n,v} = \alpha_{n,v} + \sum_{\phi_m \in \Phi} \sum_{\tau_m \in \textsc{Sat}(\phi_m)} C_v(\tau_m \langle X_n[\phi_m] \rangle) \cdot q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m) \tag{18}$$

Equations (15) and (18) represents the central result of this section: for any DFL program, we can build a variational distribution $q(\Theta, \mathbb{X} \mid \Lambda', \mathbb{M})$ and apply Equations (15) and (18) to iteratively minimize the variational KL-divergence between q and the program's posterior. In the literature [12] this paradigm goes by the name of Coordinate Ascent Variational Inference (CAVI). The equations admit a straightforward optimization: for any subset Φ' of fully exchangeable constraints, such as ϕ_2 and ϕ_4 in Equation (10), it is sufficient to use a single local parameter in $\boldsymbol{\Lambda'}$. Equation (18) can be simplified accordingly.

3.1 Stochastic Variational Inference

CAVI requires processing all the constraints in a theory before updating the global variational parameters. This can be inefficient for very large theories. To address this limitation, we follow the well-established approach of Stochastic Variational Inference (SVI) [59], and develop a method to optimize the Evidence Lower Bound (ELBO) using stochastic gradient descent. Our adaptation is presented in Algorithm 1. In a nutshell, SVI applies CAVI to small, randomly selected batches of constraints, that we denote as Φ_{batch}. For each batch, it maintains a set of intermediate global parameters $\hat{\mathbb{M}}$, optimizes them with respect to the constraints from the batch (lines 5-10), and then integrates the update into the global parameters $\mathbb{M}$ (lines 11-12). This integration is performed via a weighted average using a step size $\rho_t = (t + \tau)^{-\kappa}$ at iteration t [94], which guarantees convergence under appropriate conditions. The parameter $\tau > 0$ introduces a delay to stabilize early updates, while the forgetting rate $\kappa \in (0.5, 1]$ controls the decay of the step size over time. At line 7 the algorithm applies Equation (15) to update the local parameters $\boldsymbol{\Lambda'}$. At line 9, it applies the following

Algorithm 1: Stochastic Variational Inference for DFL PP

Input: A well-formed probabilistic program $(\mathcal{G}, \Phi)$

```
1  Randomly initialize 𝕄;
2  repeat
3      for Φ_batch ∈ Φ do
4          set 𝕄̂ = 𝕄
5          repeat
6              for φ_m ∈ Φ_batch do
7                  Compute the local parameters λ'_m
8              for μ̂_n ∈ 𝕄̂ do
9                  Compute the intermediate global parameters μ̂_n
10         until convergence of Λ' and 𝕄̂;
11         for μ_n ∈ 𝕄 do
12             μ_n := (1 − ρ_t)μ_n + ρ_t μ̂_n;
13 until convergence
```

equation to update the intermediate global parameters $\hat{\mathbb{M}}$

$$\hat{\mu}_{n,v} = \alpha_{n,v} + s \cdot \sum_{\phi_m \in \Phi_{\text{batch}}} \sum_{\tau_m \in \text{SAT}(\phi_m)} C_v(\tau_m \langle X_n[\phi_m] \rangle) \cdot q'_{\phi_m}(\tau_m \mid \boldsymbol{\lambda}'_m) \tag{19}$$

In Equation (19) the scaling factor s is the ratio between the number of constraints that mention die $\mathcal{D}_n$ in the entire theory and in the current batch.

4 From Propositional Constraints to Datalog Constraints

In the previous sections, we reviewed DFL, a propositional logic language designed for representing probabilistic programs. Additionally, we introduced a novel compilation technique for constructing variational approximations to the posterior distributions induced by DFL programs. While this method is theoretically sound, its practical applicability is limited for two main reasons: (i) real-world DFL programs may contain an extremely large number of constraints (see [1] for examples), and (ii) each constraint can admit a vast number of satisfying assignments, resulting in an exponentially large set of variational parameters in Λ'. To address the first limitation, the authors of [1] proposed using a restricted variant of Probabilistic Programming Datalog (PPDL, [5]) to encode large DFL theories in a compact representation. Their approach reformulates DFL constraints as lineage expressions [54, 63] generated by a PPDL program. In this work, we adopt the same strategy. To mitigate the second limitation, we employ knowledge compilation techniques [27] to obtain a compact representation of the set $\text{SAT}(\phi_m)$ for each constraint ϕ_m in a DFL theory. This representation is derived directly from the underlying PPDL program and enables a substantial reduction in the number of local variational parameters (i.e., the cardinality of Λ') required for inference. This contribution is novel with respect to the approach presented in [1].

Program 1 illustrates a PPDL program that admits a DFL lineage. We will use it as an example and refer the reader to [1, 5] for a complete discussion of the syntax and semantics of PPDL and its usage in the context of DFL. The predicates `dt/3`, `lp/3`, `obs/3`, and `sample/2` have a predefined semantics, which is formalized in lines 1 and 2. The predicate `dt/3` is used to declare a Pólya die: its first argument specifies a unique identifier for the die, the second defines its domain, and the third encodes the corresponding hyperparameters. The predicate `lp/3` links each Pólya die (via its identifier in the first argument) to its domain (second argument) and to a vector of latent parameters (third argument). The probabilistic Datalog clause at line 1 specifies that for every Pólya

Program 1: CoinFlippingGame

```
1  lp(Id, D, P ∈ S_|D| ~ Dir[[H]]) ← dt(Id, D, H).
2  obs(Id, R, v ∈ D ~ Cat[[P]]) ← lp(Id, D, P), sample(Id, R).
3  dt(a, [head, tail], [1, 1000]).
4  dt(b, [head, tail], [1, 100]).
5  dt(c, [head, tail], [1, 10]).
6  dt(d, [head, tail], [1, 1]).
7  dt(e, [head, tail], [10, 1]).
8  dt(f, [head, tail], [100, 1]).
9  dt(g, [head, tail], [1000, 1]).
10 round(0, tail).
11 round(1, tail).
12 round(2, head).
13 round(3, head).
14 sample(a, R) ← round(R, _).
15 sample(b, R) ← round(R, _).
16 sample(c, R) ← round(R, _).
17 hwb(1, R) ← obs(a, R, tail), obs(b, R, head), obs(c, R, head).
18 hwb(2, R) ← obs(a, R, tail), obs(b, R, tail), obs(c, R, head).
19 hwb(2, R) ← obs(a, R, head), obs(b, R, tail), obs(c, R, tail).
20 hwb(3, R) ← obs(a, R, tail), obs(b, R, tail), obs(c, R, tail).
21 order(d, e, g, R) ← hwb(1, R).
22 order(e, f, g, R) ← hwb(2, R).
23 order(f, d, g, R) ← hwb(3, R).
24 sample(C1, R) ← order(C1, C2, C3, R).
25 sample(C2, R) ← obs(C1, R, head), order(C1, C2, C3, R).
26 sample(C3, R) ← obs(C1, R, tail), order(C1, C2, C3, R).
27 win(R, O) ← obs(C2, R, O), order(C1, C2, C3, R), round(R, O).
28 win(R, O) ← obs(C3, R, O), order(C1, C2, C3, R), round(R, O).
29 qa*(R, O) ← win(R, O).
```

die declared via `dt/3`, the program samples a latent parameter vector from the appropriate Dirichlet distribution, storing the result using the `lp/3` predicate. The predicate `sample/2` is used to simulate a roll of a Pólya die: the first argument identifies the die to be rolled, while the second assigns a unique identifier to the outcome. The predicate `obs/3` records the outcome of a die roll: its first argument refers to the die, the second to the observation identifier, and the third to the observed value. The relationship between `sample/2` and `obs/3` is captured by the probabilistic clause at line 2: each occurrence of `sample/2` triggers the sampling of a value from a Categorical distribution, defined over the die's domain and parameterized by the die's vector of latent parameters. The resulting value is then associated with the generating die using the `obs/3` predicate.

With the exception of lines 1 and 2, the remainder of Program 1 consists of standard, deterministic Datalog clauses that comply with the syntactic restrictions outlined in [1]: the predicate `dt/3` is used exclusively in ground facts, while `obs/3` appears only within clause bodies. They describe the generative process encoded in the program. Predicates marked with an asterisk, such as `qa/2`, are designated as constrained; they define the conditioning expression used to compute the posterior distribution over the Pólya dice. In the case of Program 1, this condition identifies all executions of

the generative process that yield the complete set of ground instances of the predicate `qa/2` that can be derived with positive probability.

Program 1 encodes a simple coin-flipping game. A *Pólya coin* is a Pólya die with a binary domain (e.g., {head, tail}). At lines 3–9, seven Pólya coins are instantiated, labeled with the letters a through g. Coin d is fair, while the coins preceding it are progressively biased toward the outcome "tail", and those following it are progressively biased toward "head". The game is played over four rounds, and the predicate `round/2`, used at lines 10–13, associates each round with its expected outcome ("tail","tail","head","head", respectively). At each round R, coins a, b, and c are flipped, as specified in lines 14–16. Lines 17–20 implement the *hidden weighted bit* [17] function; the fact $\texttt{hwb}(n, R)$ is derived if and only if exactly n of the three coins return the value "tail", and the n-th coin (in order a, b, c) is among those returning "tail". Depending on the value of n, the program selects three out of the four remaining coins and defines an ordering over them, as specified in lines 21–23. Let (C_1, C_2, C_3) denote the selected ordering. The generative process then proceeds by flipping coin C_1. If the outcome is "head", the program flips coin C_2; otherwise, it flips coin C_3 (lines 24–26). The outcome of this final coin flip determines the result of the current round R, which is constrained, via lines 27–28, to match the target outcome specified by the predicate `round/2`.

Figure 2 illustrates the branching program obtained by grounding Program 1 with respect to a specific instance of the predicate $\texttt{round}(R, O)$ (that is, for a given round R of the game with an expected outcome O). The grounding is derived as follows: we first compute the least Herbrand model of the program, ignoring the probabilistic clauses at lines 1–2. If the resulting propositional theory contains ground instances of the predicate `sample/2`, a separate branch is created for each possible outcome of the corresponding dice rolls. For each such branch, we append the associated ground instances of the predicate `obs/3` and reiterate the derivation process. Branches that exhaust all the possible ground instances of the predicate `sample/2` without satisfying the program's constraints are pruned. Note that each directed path in Figure 2, from the root to the sink node, represents a plausible evolution of the generative process encoded by the program, i.e. an *execution path*. Moreover, each path uses any given die at most once; distinct paths may invoke the Pólya dice in different orders, and certain dice may not appear in every execution path. More precisely, each path induced by Program 1 involves exactly five out of the seven available dice. Following the terminology introduced in [84], we refer to the dice that appear in some execution paths but not in others as *volatile*. Whenever a volatile die is used along a particular path, we say that the die is *active* on that path. In the case of Program 1, the Pólya coins $\{d, e, f, g\}$ are all volatile.

To obtain a DFL theory from Program 1, we proceed by deriving the lineage expressions of its ground predicates. A ground predicate is one where all variables have been replaced with

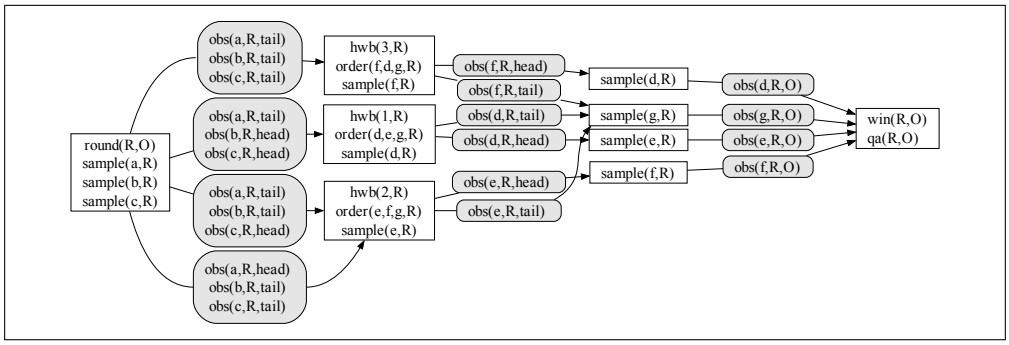


Fig. 2. Grounding of Program 1, parameterized by the predicate $\texttt{round}(R, O)$.

$$
\begin{aligned}
\phi\,[\text{round}(R,O)\,] &:= \top \\
\phi\,[\text{sample}(C,R)\,] &:= \phi[\text{round}(R,O)\,]\ \ \forall C \in \{a,b,c\} \\
\phi\,[\text{hwb}(1,R)\,] &:= \phi[\text{obs}(a,R,\text{tail})\,] \wedge \phi[\text{obs}(b,R,\text{head})\,]\wedge \\
&\quad \wedge \phi[\text{obs}(c,R,\text{head})\,] \\
\phi\,[\text{hwb}(2,R)\,] &:= (\phi[\text{obs}(a,R,\text{tail})\,] \wedge \phi[\text{obs}(b,R,\text{tail})\,]\wedge \\
&\quad \wedge \phi[\text{obs}(c,R,\text{head})\,])\vee \\
&\quad \vee (\phi[\text{obs}(a,R,\text{head})\,] \wedge \phi[\text{obs}(b,R,\text{tail})\,]\wedge \\
&\quad \wedge \phi[\text{obs}(c,R,\text{tail})\,]) \\
\phi\,[\text{hwb}(3,R)\,] &:= \phi[\text{obs}(a,R,\text{tail})\,] \wedge \phi[\text{obs}(b,R,\text{tail})\,]\wedge \\
&\quad \wedge \phi[\text{obs}(c,R,\text{tail})\,] \\
\phi\,[\text{order}(d,e,g,R)\,] &:= \phi\,[\text{hwb}(1,R)\,] \\
\phi\,[\text{order}(e,f,g,R)\,] &:= \phi\,[\text{hwb}(2,R)\,] \\
\phi\,[\text{order}(f,d,g,R)\,] &:= \phi\,[\text{hwb}(3,R)\,] \\
\phi\,[\text{sample}(d,R)\,] &:= \phi\,[\text{order}(d,e,g,R)\,]\vee \\
&\quad \vee (\phi\,[\text{obs}(f,R,\text{head})\,] \wedge \phi\,[\text{order}(f,d,g,R)\,]) \\
\phi\,[\text{sample}(e,R)\,] &:= \phi\,[\text{order}(e,f,g,R)\,]\vee \\
&\quad \vee (\phi\,[\text{obs}(d,R,\text{head})\,] \wedge \phi\,[\text{order}(d,e,g,R)\,]) \\
\phi\,[\text{sample}(f,R)\,] &:= \phi\,[\text{order}(f,d,g,R)\,]\vee \\
&\quad \vee (\phi\,[\text{obs}(e,R,\text{head})\,] \wedge \phi\,[\text{order}(e,f,g,R)\,]) \\
\phi\,[\text{sample}(g,R)\,] &:= (\phi\,[\text{obs}(d,R,\text{tail})\,] \wedge \phi\,[\text{order}(d,e,g,R)\,])\vee \\
&\quad \vee (\phi\,[\text{obs}(f,R,\text{tail})\,] \wedge \phi\,[\text{order}(f,d,g,R)\,])\vee \\
&\quad \vee (\phi\,[\text{obs}(e,R,\text{tail})\,] \wedge \phi\,[\text{order}(e,f,g,R)\,]) \\
\phi\,[\text{win}(R,O)\,] &:= \phi[\text{round}(R,O)\,]\wedge \\
&\quad \wedge ((\phi\,[\text{obs}(e,R,O)\,] \wedge \phi\,[\text{order}(d,e,g,R)\,])\vee \\
&\quad \vee (\phi\,[\text{obs}(g,R,O)\,] \wedge \phi\,[\text{order}(d,e,g,R)\,])\vee \\
&\quad \vee (\phi\,[\text{obs}(d,R,O)\,] \wedge \phi\,[\text{order}(f,d,g,R)\,])\vee \\
&\quad \vee (\phi\,[\text{obs}(g,R,O)\,] \wedge \phi\,[\text{order}(f,d,g,R)\,])\vee \\
&\quad \vee (\phi\,[\text{obs}(f,R,O)\,] \wedge \phi\,[\text{order}(e,f,g,R)\,])\vee \\
&\quad \vee (\phi\,[\text{obs}(g,R,O)\,] \wedge \phi\,[\text{order}(e,f,g,R)\,])) \\
\phi\,[\text{qa}(R,O)\,] &:= \phi\,[\text{win}(R,O)\,] \\
\phi\,[\text{obs}(C,R,\text{head})\,] &:= (C[R] = \text{head}) \wedge \phi\,[\text{sample}(C,R)\,]\ \ \forall C \in \{a,\ldots,g\} \\
\phi\,[\text{obs}(C,R,\text{tail})\,] &:= (C[R] = \text{tail}) \wedge \phi\,[\text{sample}(C,R)\,]\ \ \forall C \in \{a,\ldots,g\}
\end{aligned}
$$

Fig. 3. The lineage of Program 1, parameterized by round(R,O).

concrete values (e.g., `sample(a,0)` rather than `sample(C,R)`). If $p(\cdot)$ is a ground predicate then we denote by $\phi[p(\cdot)]$ its lineage expression. We follow the conventions introduced in [1]: the lineage of deterministic facts is defined as $\top$, while the lineage of inferred facts is computed using the standard rules [54], except for ground facts of the form obs(d,r,v), whose lineage is defined as $(d[r] = v) \wedge \phi[\text{sample}(d,R)]$. Figure 3 illustrates the lineage expressions derived from Program 1. Note that the expressions $\phi[\text{qa}(0,\texttt{tail})]$ and $\phi[\text{qa}(1,\texttt{tail})]$ represent finitely exchangeable events, as do $\phi[\text{qa}(2,\texttt{head})]$ and $\phi[\text{qa}(3,\texttt{head})]$. Taken together, these four expressions form a set of partially exchangeable events and constitute a well-defined DFL program. Furthermore, for any volatile die, the lineage of its associated `sample/2` ground predicate identifies all the execution

paths in which the die is active. Following the terminology of [84], we refer to the disjuncts in this lineage expression as the *activation conditions* of the die.

Clearly, not all PPDL programs generate a DFL lineage. In addition to adhering to the syntactic restrictions and key constraints described in [1], a sufficient condition for a PPDL program to admit a DFL lineage is the presence of a special *partitioning attribute* R with the following properties: (i) R serves as a key for the constrained relation (`qa`), and (ii) R determines the second attribute of every tuple in the relation `sample`.

The next step in analyzing Program 1 is to identify a compact representation of its lineage expressions, with the goal of minimizing the number of local variational parameters in Λ' required for inference. To this end, we adapt well-established formalisms from the field of knowledge compilation [30] to our setting. Following [30], we represent the constraints in a DFL program as Boolean circuits, i.e. rooted directed acyclic graphs (DAGs) in which internal nodes are labeled with Boolean operators, and sink nodes are labeled with atomic constraints or constant symbols. Throughout this section, we use the terms "circuits" and "expressions" interchangeably. Note that our definition of a circuit deviates from the classical one, which assumes that variables range over Boolean domains, whereas Pólya dice do not necessarily respect such restriction. From [30] we adopt the operators *independent conjunction* ($\odot$) and *deterministic disjunction* ($\oplus$). We denote by $\phi_m \odot \phi_{m'}$ the logical conjunction of two expressions that do not share any variables (i.e., $\textsc{Var}(\phi_m) \cap \textsc{Var}(\phi_{m'}) = \emptyset$), and by $\phi_m \oplus \phi_{m'}$ the logical disjunction of two expressions that are mutually exclusive (i.e., $\textsc{Sat}(\phi_m, \textsc{Var}(\phi_m, \phi_{m'})) \cap \textsc{Sat}(\phi_{m'}, \textsc{Var}(\phi_m, \phi_{m'})) = \emptyset$).

Similarly to [30], we denote by `NNF` the class of all the circuits that are free of negation, except for their atoms, and by `d-DNNF` the subclass of `NNF` circuits that use only the operators $\odot$ and $\oplus$. For any expression ϕ_m, we denote by $(\phi_m || X_n[\phi_m] \mapsto v)$ the expression obtained by replacing every atomic constraint in ϕ_m that mentions variable $X_n[\phi_m]$ with either the constant $\bot$, when the constraint and $X_n[\phi_m] = v$ are mutually exclusive, or the constant $\top$, otherwise. Notice that by construction $X_n[\phi_m] \notin \textsc{Var}(\phi_m || X_n[\phi_m] \mapsto v)$. The *Boole/Shannon decomposition* [16, 100] uses the conditioning operator $(\phi_m || \cdot)$ to factorize any circuit ϕ_m with respect to one of its variables as follows:

$$\phi_m \equiv \bigoplus_{v \in \boldsymbol{V}_n} ((X_n[\phi_m] = v) \odot (\phi_m || X_n[\phi_m] \mapsto v))$$

We say that $X_n[\phi_m] \in \textsc{Var}(\phi_m)$ is *inessential* in ϕ_m if there exists another expression ϕ'_m that is logically equivalent to ϕ_m and does not use $X_n[\phi_m]$. Otherwise, we say that $X_n[\phi_m]$ is *essential* in ϕ_m. Notice that $X_n[\phi_m]$ is inessential in ϕ_m if and only if $(\phi_m || X_n[\phi_m] \mapsto v)$ is logically equivalent to $(\phi_m || X_n[\phi_m] \mapsto v')$ for any two v and v' in $\boldsymbol{V}_n$. We can generalize the conditioning operator to handle multiple variables. If $\mathbb{X}' \subseteq \textsc{Var}(\phi_m)$ and τ is a term expression in $\textsc{Asst}(\mathbb{X}')$, we denote by $(\phi_m || \tau)$ the expression obtained by repeatedly applying the $(\phi_m || \cdot)$ operator to map each variable in $\mathbb{X}'$ to the corresponding value specified by τ.

Let $\Psi = \{\psi_1, \ldots, \psi_P\}$ be a collection of constraints that form a partition of $\textsc{Asst}(\textsc{Var}(\Psi))$ (i.e. all the constraints are satisfiable, mutually exclusive and their disjunction is a tautology), with $\textsc{Var}(\Psi) \subset \textsc{Var}(\phi_m)$. We say that Ψ forms a *sentential partition* [27] of ϕ_m when there exists a set of expressions $\Xi = \{\chi_1, \ldots, \chi_P\}$ such that no two expressions in Ξ are logically equivalent and

$$\phi_m \equiv \bigoplus_{\psi_p \in \Psi} \left(\psi_p \odot \chi_p\right) \tag{20}$$

The above equivalence is satisfied when $\chi_p \equiv (\phi_m || \tau)$ for every τ in $\textsc{Sat}\left(\psi_p, \textsc{Var}(\psi_p)\right)$ and every ψ_p in Ψ. Reusing the terminology from [27], we refer to $\{\psi_1, \ldots, \psi_P\}$ and $\{\chi_1, \ldots, \chi_P\}$ as the *primes*

and the *subs* of the partition, respectively, and to $\textsc{Var}(\Psi)$ and $\textsc{Var}(\Xi)$ as the *prime-* and *sub variables* of the partition, respectively.

Definition 1. *Let $\Psi = \{\psi_1, \ldots, \psi_P\}$ be a collection of constraints that are all satisfiable, mutually exclusive and such that $\textsc{Var}(\Psi) \subset \textsc{Var}(\phi_m)$ and $\phi_m \models \left(\vee_{\psi_p \in \Psi} \psi_p\right)$. If there exists a set of expressions $\Xi = \{\chi_1, \ldots, \chi_P\}$ such that no two expressions in Ξ are logically equivalent and Equation* (20) *holds true, then we say that Ψ forms a* positive sentential decomposition *of ϕ_m.*

Note that we can obtain a positive sentential decomposition from a sentential partition by simply dropping the prime associated with the sub $\bot$, if any.

Definition 2. *We denote by* upSDD *(acronym for "unstructured positive SDD") the subclass of* d-DNNF *circuits that use the $\odot$ and $\oplus$ operators exclusively within positive sentential decompositions.*

The DFL lineage generated by a ground PPDL theory can be naturally represented using upSDD circuits. To construct such a representation, it suffices to apply a positive sentential decomposition at each branching point in the ground program. The dice being rolled at the branching point serve as the prime variables in the decomposition, while all other variables play the role of sub variables. Figure 4 illustrates the upSDD circuit derived from Program 1. In the figure, dashed edges point at prime expressions, while solid edges point at sub expressions. Note that whenever a volatile die remains inactive over a certain branch of the program, its associated variable remains inessential in the corresponding sub expression in the circuit.

For any d-DNNF circuit ϕ_m, we denote by $\textsc{DSat}(\phi_m)$ the set of term expressions returned by Algorithm 2. Note that each such term satisfies ϕ_m, but does not necessarily use all the variables in $\textsc{Var}(\phi_m)$. Furthermore, all the terms are mutually exclusive and for any term $\tau \in \textsc{Sat}(\phi_m, \textsc{Var}(\phi_m))$ there exists a term $\tau' \in \textsc{DSat}(\phi_m)$ such that $\tau \models \tau'$. Thus, we can conclude that $\textsc{DSat}(\phi_m)$ encodes a partition of $\textsc{Sat}(\phi_m, \textsc{Var}(\phi_m))$. Note that Algorithm 2 is not guaranteed to produce the same output when applied to two distinct d-DNNF circuits that are logically equivalent.

If $\Phi = \{\phi_1, \ldots, \phi_M\}$ is a collection of partially exchangeable d-DNNF circuits, we denote by $\textsc{DSat}(\Phi)$ the set $\{(\tau_1 \wedge \ldots \wedge \tau_M) \mid (\tau_1, \ldots, \tau_M) \in \times_{\phi_m \in \Phi} \textsc{DSat}(\phi_m)\}$. By construction, $\textsc{DSat}(\Phi)$ defines a partition of $\textsc{Sat}(\Phi, \textsc{Var}(\Phi))$ and may be exponentially smaller in size. To take advantage of this, we can reformulate Equation (12) as follows:

$$p(\Theta \mid \Phi, \mathbb{A}) = \sum_{\tau \in \textsc{DSat}(\Phi)} p(\Theta \mid \tau, \mathbb{A}) \cdot P(\tau \mid \Phi, \mathbb{A}) \tag{21}$$

Starting from Equation (21) we can derive a variational approximation of the posterior $p(\Theta \mid \Phi, \mathbb{A})$ as per Section 3, and obtain a model with, potentially, a much smaller set of local parameters (Λ'). Figure 5 shows the set of terms obtained by executing Algorithm 2 on the circuit presented in Figure 4. Note that each term identifies a path in the ground program from Figure 2, and only uses variables that are active along that path.

5 Implementation

We implemented the variational inference algorithm presented in this paper as an extension to the StarfishDB engine, offering an alternative to the original collapsed Gibbs sampler introduced in [1]. We refer to the two approaches as *SFDB-SVI* and *SFDB-CGS*, respectively. Our implementation adheres to the design principles of StarfishDB: the engine leverages just-in-time (JIT) compilation [42] to optimize the inference methods and uses the Acero relational query processor from Apache Arrow [46] to perform grounding. This enables the use of push-based [99] physical query plans, columnar storage [14] and vectorized execution [15] during the process.

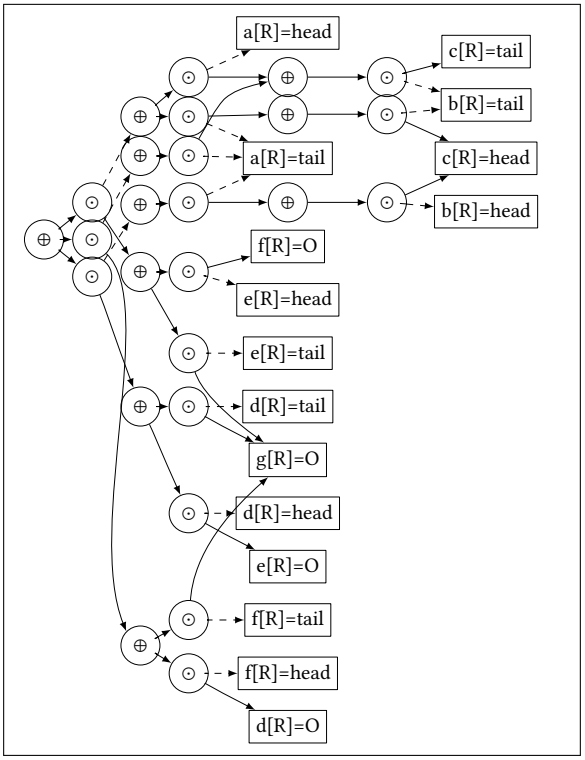


Fig. 4. The lineage of Program 1 represented as an upSDD circuit, parameterized by predicate round(R, O). Unary applications of the $\oplus$ operator are treated as no-operations (no-ops).

Algorithm 2: DSat

Input: A d-DNNF circuit ϕ_m
Output: A set of term expressions that satisfy ϕ_m

if $\phi_m = \top$ **then**
  return $\{\top\}$
else if ϕ_m *is an atomic constraint in the form* $X_n[\phi_m] \in \boldsymbol{V}'_n$ **then**
  return $\textsc{Sat}\,(\phi_m, \{X_n[\phi_m]\})$
else if $\phi_m = \phi \odot \phi'$ **then**
  return $\{\tau \odot \tau' \mid \tau \in \textsc{DSat}\,(\phi)\,, \tau' \in \textsc{DSat}\,(\phi')\}$
else if $\phi_m = \phi \oplus \phi'$ **then**
  return $\textsc{DSat}\,(\phi) \cup \textsc{DSat}\,(\phi')$
else
  return $\emptyset$; // $\phi_m = \bot$

In StarfishDB, expressions like those shown in Figures 4 and 5 are referred to as *expression templates*, due to their use of parameters (e.g., the parameters R and O in the example discussed in this paper). Through JIT compilation, the engine translates these expression templates into efficient inference methods that take as input a valuation of the parameters and execute a single basic inference step as output. The expression templates are derived directly from the Datalog specification of the program, without using propositional optimization tools such as [29, 72, 87].

$$
\begin{array}{l}
(a[R]{=}\ \text{tail}) \wedge (b[R]{=}\ \text{tail}) \wedge (c[R]{=}\ \text{tail}) \wedge (d[R]{=}\ \text{head}) \wedge (e[R] = O) \\
(a[R]{=}\ \text{tail}) \wedge (b[R]{=}\ \text{tail}) \wedge (c[R]{=}\ \text{tail}) \wedge (d[R]{=}\ \text{tail}) \wedge (g[R] = O) \\
(a[R]{=}\ \text{head}) \wedge (b[R]{=}\ \text{tail}) \wedge (c[R]{=}\ \text{tail}) \wedge (e[R]{=}\ \text{head}) \wedge (f[R] = O) \\
(a[R]{=}\ \text{head}) \wedge (b[R]{=}\ \text{tail}) \wedge (c[R]{=}\ \text{tail}) \wedge (e[R]{=}\ \text{tail}) \wedge (g[R] = O) \\
(a[R]{=}\ \text{tail}) \wedge (b[R]{=}\ \text{tail}) \wedge (c[R]{=}\ \text{head}) \wedge (e[R]{=}\ \text{head}) \wedge (f[R] = O) \\
(a[R]{=}\ \text{tail}) \wedge (b[R]{=}\ \text{tail}) \wedge (c[R]{=}\ \text{head}) \wedge (e[R]{=}\ \text{tail}) \wedge (g[R] = O) \\
(a[R]{=}\ \text{tail}) \wedge (b[R]{=}\ \text{head}) \wedge (c[R]{=}\ \text{head}) \wedge (f[R]{=}\ \text{head}) \wedge (d[R] = O) \\
(a[R]{=}\ \text{tail}) \wedge (b[R]{=}\ \text{head}) \wedge (c[R]{=}\ \text{head}) \wedge (f[R]{=}\ \text{tail}) \wedge (g[R] = O)
\end{array}
$$

Fig. 5. The terms generated by executing Algorithm 2 on the circuit depicted in Figure 4.

This is because probabilistic Datalog programs often induce a natural die-rolling ordering over the Pólya dice, which translates directly into a reasonable upSDD decomposition. Furthermore, the dice often have non-Boolean domains, whereas standard d-DNNF expressions are typically restricted to Boolean variables.

In the case of *SFDB-CGS*, the engine translates a circuit like the one depicted in Figure 4 into a function that samples a term from the corresponding DSAT and uses it to perform one step in the Markov chain generated by a collapsed Gibbs sampling scheme. In the case of *SFDB-SVI*, the engine translates an expression like the one shown in Figure 5 (obtained by taking a disjunction of all the terms in DSAT) into a method that updates the variational parameters of the model by applying Equations (15) and (19) as per Algorithm 1. The grounding engine simply iterates over all the constraints in Φ, invoking the JIT-compiled method for each constraint with the appropriate values for the template's parameters. Thanks to this approach, the engine compiles a single inference method for each expression *template*, which is then executed a large number of times, thereby amortizing the compilation overhead. All the probabilistic programs presented in this paper and in [1] require the compilation of only a single expression template. Furthermore, the engine maximizes efficiency through the diligent use of tight loops, parallelization and batched memory allocation.

6 Experiments

We compare our framework against two baselines: *MALLET* [82] and *ProbLog* [68]. MALLET is a general-purpose NLP library that includes a highly optimized implementation of Latent Dirichlet Allocation (LDA, [13]), a probabilistic model for topic extraction. We compare it against a DFL implementation of the model, using StarfishDB to derive a variational inference method and a collapsed Gibbs sampler from it. ProbLog is a general-purpose PP framework, that adopts a Datalog-like language and relies on exact inference (in the form of variable elimination). For comparison with StarfishDB, we reproduce in DFL one of the canonical programs from ProbLog's official tutorial, a simplified Ising model. All experiments are run on a 32-core Xeon (2.10GHz) machine with 512GB of RAM, running Red Hat Enterprise Linux version 8.1.

Experiment 1: Latent Dirichlet Allocation. The PPDL/DFL formulation of LDA that we use here is given in [1, 84]. We apply the techniques presented in this paper to compile a variational inference method for it. LDA is an excellent choice for our purpose, as a variety of specialized inference methods have been proposed for it. These include both variational inference methods [13, 45, 58, 59, 109] and Monte Carlo methods [55], alongside well-established performance metrics [3, 117] and a wide range of publicly available datasets. Our benchmark compares four different inference methods for LDA: (1) *SFDB-SVI*, our proposed approach; (2) *SFDB-CGS*, the collapsed Gibbs sampler derived from the PPDL formulation of LDA using the compilation techniques proposed in

Dataset	Timestamp	SFDB-SVI	SFDB-CGS	C++SVI	Mallet
PubMed (100 topics)	T1 (773 s)	**-4.0673**	−4.0767	−4.0685	−4.0736
	T2 (971 s)	**-4.0664**	−4.0733	−4.0679	−4.0690
Wikipedia (200 topics)	T1 (1659 s)	**-5.5353**	−5.6240	−5.6567	−5.7077
	T2 (3281 s)	**-5.5362**	−5.6173	−5.6099	−5.6281

Note: Log-likelihood values are scaled by 10^8.

Table 1. Log-likelihood evaluated at timestamps *T1* and *T2* for PubMed and Wikipedia datasets. Higher log-likelihood values are more desirable.

[1]; (3) *C++SVI*, a C++ implementation of [58], developed by us; and (4) *MALLET* [82], a popular Java implementation of [55]. Methods (1) and (3) use variational inference, while methods (2) and (4) rely on Gibbs sampling. Methods (1) and (2) are *general-purpose*, as both are generated by a compiler that can be applied to other probabilistic models, simply by selecting the appropriate PPDL theory. In contrast, methods (3) and (4) are *special-purpose*, as they implement inference mechanisms that leverage structural properties specific to LDA.

We run our experiments on two text corpora: PubMed, which contains 8.2 million biomedical abstracts (730 million tokens, with a vocabulary size of 141,043 distinct words), and Wikipedia, with 4 million articles (1 billion tokens, with a vocabulary size of 192,000 distinct words). The vocabulary size is measured after stemming and normalizing the tokens to filter out rare and stop words. These datasets were selected for their public availability, varying scales, and widespread use in prior LDA research, providing a representative benchmark. For *SFDB-SVI* and *C++SVI*, we set the batch sizes to 4 million words for the PubMed dataset and 8.5 million words for the Wikipedia dataset. For both algorithms, we maintain consistent step size parameters: κ (forgetting rate) set to 0.7 and τ (delay) set to 1 throughout all experiments. These parameter values were based on the study in [58] and further validated through our own experimental testing. All competing methods use symmetric Dirichlet priors with a value of 0.2 for document composition and 0.1 for topic composition. For the PubMed dataset, we train LDA models with 100 topics, while for the larger Wikipedia dataset, we use 200 topics. For all datasets, we use 95% of the data for training and 5% for testing. Each competing inference algorithm is run to train an LDA model on the training documents (95%), and the quality of the resulting model is assessed by estimating the log-likelihood of the documents in the test set (5%) [23, 117]. Since computing the exact likelihood of a large LDA model is intractable [117], we use MALLET's `evaluate-topics` tool to obtain an approximate measure. The log-likelihood is measured at two specific timestamps, denoted as *T1* and *T2*. *T1* represents the time required for *SFDB-SVI* to complete a single full sweep over all the training data (773 seconds for PubMed, 1,659 seconds for Wikipedia). *T2* represents the time needed by MALLET's Gibbs sampler to perform 40 full Monte Carlo sweeps over the training data (971 seconds for PubMed, 3,281 seconds for Wikipedia). Table 1 reports our findings. To ensure a fair comparison, the timings in Table 1 do not include I/O operations or compilation. We observed StarfishDB to be more I/O efficient than *C++SVI* and MALLET, with a total I/O time, including grounding, of 56 seconds for PubMed and 70 seconds for Wikipedia. The advantage is largely due to the compressed data format used in Arrow. JIT compilation requires only 5 seconds with 100 topics and 12 seconds with 200 topics.

Analysis of the results: Overall, we observe that the variational methods converge faster than their Monte Carlo counterparts, reaching a mostly stable likelihood by timestamp *T1*. Unlike the variational methods, the Monte Carlo methods are guaranteed to converge asymptotically to

Config	Truth	SFDB-SVI		SFDB-CGS		ProbLog	
(Size)	(Prob.)	Err.	Time (s)	Err.	Time (s)	Err.	Time (s)
2×2	0.819	0.014	0.0013	0	9.4	0	0.014
3×3	0.792	0.050	0.0015	0.004	14.3	0	0.280
4×4	0.790	0.054	0.0028	0.008	50.1	0	1.070
5×5	0.793	0.008	0.0048	0	83.9	0	113.8
32×32	?	0.036*	0.2350	—	182.0	T/O	T/O
64×64	?	0.030*	0.3900	—	223.7	T/O	T/O

* Relative error w.r.t. SFDB-CGS.

Table 2. Ising model performance. Error measures absolute difference from ground truth probability that corner sites match. Ground truth: exact enumeration for $\leq 5 \times 5$, SFDB-CGS for larger grids (*). Times in seconds; T/O = timeout.

the true posterior, and thus continue to improve the model's likelihood until timestamp *T2*. Our experiments are designed to address two key questions: (1) Can a PPDL program generate an inference method that competes with the performance of a hard-coded implementation? Achieving this would be highly desirable, as the PPDL formulation of LDA, which consists of just five Datalog clauses, is significantly simpler and more compact than the hundreds of lines of C++ code typically required for implementing a finely tuned inference method for LDA. (2) Is our new variational inference method any better than the existing Monte Carlo method offered by StarfishDB?

The answer to the first question is *yes*: the LDA models generated by *SFDB-SVI* and *SFDB-CGS* are largely comparable to those produced by specialized tools like *C++SVI* and MALLET, both in terms of quality and training cost. Using PPDL eliminates the need for specialized expertise in the implementation of optimization algorithms, making the language more accessible to the average database practitioner and easy to integrate into relational database systems. The answer to the second question is more nuanced. Our findings align with previous experiences reported in the literature [12]: both variational and Monte Carlo methods have their respective advantages and drawbacks. While both approaches are practical and widely used, variational methods generally converge faster and are easier to parallelize, offering better scalability. On the other hand, Monte Carlo methods remain the preferred choice when asymptotic optimality is required or when the use of small-scale data does not justify the development of a variational method.

Experiment 2: Ising Model. In this experiment, we replicate a categorical Markov Random Field (MRF) of arity two from the official ProbLog tutorial[1]. This MRF implements a simplified Ising model [70]. We model a grid of $r \times r$ sites $\mathbb{S} = \{s_1, s_2, \ldots, s_{r^2}\}$, where each site takes binary values $\{0, 1\}$. Each site interacts with its four immediate neighbors (up, down, left, right). We denote the set of all neighboring pairs by $\mathcal{N}$. The joint distribution $p(\mathbb{S} \mid a, b) \propto \prod_{\langle s_i, s_j \rangle \in \mathcal{N}} \psi(s_i, s_j)$ factorizes over neighbor pairs through potential function $\psi(s_i, s_j)$ that assigns weight a when neighbors match and weight b when they differ. To encode this model in DFL, we instantiate a binary Pólya urn X_i with colors $\{0, 1\}$ for each site s_i. We encode MRF potentials through constraint repetition: for each neighboring pair $\langle s_i, s_j \rangle \in \mathcal{N}$, we create multiple matching constraints (enforcing that urns X_i and X_j draw the same color) and unmatching constraints (enforcing different colors). The ratio of matching to unmatching constraints equals the ratio of potential weights $a : b$. We set $a = 0.8$ and $b = 0.2$, scaling by 100 to obtain 80 matching and 20 unmatching constraints per neighbor pair. We measure the probability that corner sites share the same state, validating *SFDB-SVI* against

[1] https://dtai.cs.kuleuven.be/problog/tutorial/various/04_nampally.html

exact C++ solver, ProbLog, and *SFDB-CGS* on grids from 2×2 to 5×5, plus 32×32 and 64×64 for scalability.

Analysis of the results: Table 2 shows exact methods time out beyond 5×5 (2^{25} configurations), while *SFDB-SVI* maintains errors below 0.054 with orders of magnitude speedup compared to *SFDB-CGS*. While exact inference is preferable when computationally feasible, many models including Ising and LDA make it intractable at scale, demonstrating the practical utility of approximate inference.

7 Related Work

Datalog in Probabilistic Programming: We refer readers to [1] for a detailed contextualization of StarfishDB in relation to other competing probabilistic programming frameworks. Two of the frameworks that are most similar are PPDL [5] and ProbLog [39, 114]. The language used in StarfishDB is a fragment of PPDL, whereas ProbLog utilizes Sato's distributional semantics [79, 96]. PPDL is not equipped with any inference method, while ProbLog typically relies on exact inference through weighted model counting. Interestingly, there is a variant of ProbLog that incorporates Bayesian priors [21]. Our line of work fits within the broader field of Statistical Relational Learning [92], which has led to the development of several probabilistic programming languages [5, 48, 68, 85, 91, 93, 97]. These languages utilize first-order logic to efficiently represent large sets of constraints [113]. The authors of [19, 65] advocate for the use of database systems as an effective computational backbone for inference and machine learning; our work adheres to this perspective.

Variational Inference in Probabilistic Programming: Variational Inference is widely used in Bayesian statistics [6, 11, 12, 44, 47, 59, 71] and implemented in frameworks like Pyro [9], Edward [112], ProbTorch [108] and Turing.jl [43], Gen system [7, 25]. Some probabilistic databases use exact circuit-based inference [98] or sampling approximations [61]. Unlike common applications of variational inference, our approach is tailored to DFL theories and frees the user from providing a guide function. Systems like GenSQL [62] and MultiPPL [107] abstract inference via a unified interface that admits external PPLs/engines as backends, enabling users to focus on program expressivity rather than implementing inference.

Knowledge Compilation for Probabilistic Inference: An excellent survey on the usage of Boolean circuits in the context of database systems [66], including probabilistic query evaluation, is given in [2]. Fairly recent advances [18, 27, 28] in knowledge compilation techniques have boosted tractable inference for SRL, typically through weighted model counting (WMC, [34, 41, 60, 86, 92]), Approximate model counting [22] and exact model counting [101]. This approach has been extended to various compilation targets like probabilistic decision graphs [64], arithmetic circuits [28], and sum-product networks [89, 105], which trade expressivity for tractability. The idea of using knowledge compilation techniques in first order logic based probabilistic programs boils down to having a lifted representation of the probabilistic program that avoids repeated calculations [35, 50, 111]. The circuit class used in this paper, upSDD, is a simplified version of the classic SDDs [27]. Unlike SDDs, upSDDs do not need to adhere to any vtree, which imposes a hierarchical partitioning scheme for the use of the $\odot$ operator. This relaxation is made because our work focuses solely on efficiently computing DSAT, while maintaining the conciseness of the representation. Note that the circuit depicted in Figure 4 does not admit any vtree. Other differences include the use of non-binary variables and the absence of sub expressions that evaluate to $\bot$. SDDs power numerous applications including exact inference tools (PySDD [83], Ace [115]), statistical relational learning systems (ProbLog2 [39], DeepProbLog [76]), graphical model compilation [24], circuit operations [80, 102], and constraint-aware PSDD frameworks [51, 69, 73, 88, 103, 104]. Arithmetic circuits have also been used to directly compute the ELBO in variational inference [105].

8 Conclusions

We introduced a novel variational inference method for StarfishDB, paired with a novel knowledge compilation technique to minimize the number of variational parameters. We implemented the new method and verified its correctness and scalability against a real-world, intractable inference task (LDA). In the future we plan to extend DFL to non-parametric models [11] and to explore the derivation of confidence bounds, in the spirit of [49, 114].

In the preparation of this manuscript, we used generative AI (ChatGPT) for spell checking, rephrasing, and grammar correction.

References

[1] Ouael Ben Amara, Sami Hadouaj, and Niccolò Meneghetti. 2024. StarfishDB: A Query Execution Engine for Relational Probabilistic Programming. *Proc. ACM Manag. Data* 2, 3 (2024), 185. doi:10.1145/3654988

[2] Antoine Amarilli and Florent Capelli. 2024. Tractable Circuits in Database Theory. *SIGMOD Rec.* 53, 2 (2024), 6–20. doi:10.1145/3685980.3685982

[3] Arthur U. Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. 2012. On Smoothing and Inference for Topic Models. *CoRR* abs/1205.2662 (2012). arXiv:1205.2662 http://arxiv.org/abs/1205.2662

[4] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2017. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *J. Mach. Learn. Res.* 18 (2017), 109:1–109:67. http://jmlr.org/papers/v18/15-631.html

[5] 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. doi:10.1145/3132700

[6] Matthew J. Beal. 2003. Variational algorithms for approximate Bayesian inference. https://api.semanticscholar.org/CorpusID:11861569

[7] McCoy R. Becker, Alexander K. Lew, Xiaoyan Wang, Matin Ghavami, Mathieu Huot, Martin C. Rinard, and Vikash K. Mansinghka. 2024. Probabilistic Programming with Programmable Variational Inference. *Proc. ACM Program. Lang.* 8, PLDI, Article 233 (June 2024), 25 pages. doi:10.1145/3656463

[8] Matthew N. Bernstein. 2022. Functionals and functional derivatives. https://mbernste.github.io. https://mbernste.github.io/posts/functionals/ Accessed: 2025-06-01.

[9] 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.

[10] David Blackwell and James B MacQueen. 1973. Ferguson distributions via Pólya urn schemes. *The annals of statistics* 1, 2 (1973), 353–355.

[11] David M. Blei and Michael I. Jordan. 2004. Variational methods for the Dirichlet process. In *Proceedings of the Twenty-First International Conference on Machine Learning* (Banff, Alberta, Canada) *(ICML '04)*. Association for Computing Machinery, New York, NY, USA, 12. doi:10.1145/1015330.1015439

[12] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* 112, 518 (2017), 859–877.

[13] 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

[14] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. 2008. Breaking the memory wall in MonetDB. *Commun. ACM* 51, 12 (2008), 77–85. doi:10.1145/1409360.1409380

[15] Peter A. Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-Pipelining Query Execution. In *Second Biennial Conference on Innovative Data Systems Research, CIDR 2005, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*. www.cidrdb.org, 225–237. http://cidrdb.org/cidr2005/papers/P19.pdf

[16] George Boole. 2021. An Investigation of the Laws of Thought on which are founded the mathematical theories of Logic and Probabilities (1854). (2021).

[17] Simone Bova. 2016. SDDs Are Exponentially More Succinct than OBDDs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, 929–935. doi:10.1609/AAAI.V30I1.10107

[18] Maddy Bowers, Alexander K. Lew, Joshua B. Tenenbaum, Armando Solar-Lezama, and Vikash K. Mansinghka. 2025. Stochastic Lazy Knowledge Compilation for Inference in Discrete Probabilistic Programs. *Proc. ACM Program. Lang.* 9, PLDI (2025), 1863–1887. doi:10.1145/3729325

[19] 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. doi:10.1145/2463676.2465283

[20] 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).

[21] Federico Cerutti, Lance M. Kaplan, Angelika Kimmig, and Murat Sensoy. 2019. Probabilistic Logic Programming with Beta-Distributed Random Variables. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 7769–7776. doi:10.1609/AAAI.V33I01.33017769

[22] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2013. A Scalable Approximate Model Counter. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8124)*, Christian Schulte (Ed.). Springer, 200–216. doi:10.1007/978-3-642-40627-0_18

[23] Jonathan D. Chang, Jordan L. Boyd-Graber, Sean Gerrish, Chong Wang, and David M. Blei. 2009. Reading Tea Leaves: How Humans Interpret Topic Models. 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., 288–296. https://proceedings.neurips.cc/paper/2009/hash/f92586a25bb3145facd64ab20fd554ff-Abstract.html

[24] Arthur Choi, Doga Kisa, and Adnan Darwiche. 2013. Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 12th European Conference, ECSQARU 2013, Utrecht, The Netherlands, July 8-10, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7958)*, Linda C. van der Gaag (Ed.). Springer, 121–132. doi:10.1007/978-3-642-39091-3_11

[25] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 221–236. doi:10.1145/3314221.3314642

[26] Nilesh N. Dalvi and Dan Suciu. 2007. Efficient query evaluation on probabilistic databases. *VLDB J.* 16, 4 (2007), 523–544. doi:10.1007/s00778-006-0004-3

[27] Adnan Darwiche. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, Toby Walsh (Ed.). IJCAI/AAAI, 819–826. doi:10.5591/978-1-57735-516-8/IJCAI11-143

[28] Adnan Darwiche. 2021. Tractable Boolean and Arithmetic Circuits. In *Neuro-Symbolic Artificial Intelligence: The State of the Art*, Pascal Hitzler and Md. Kamruzzaman Sarker (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 342. IOS Press, 146–172. doi:10.3233/FAIA210353

[29] Adnan Darwiche et al. 2004. New advances in compiling CNF to decomposable negation normal form. In *Proc. of ECAI*. Citeseer, 328–332.

[30] Adnan Darwiche and Pierre Marquis. 2002. A Knowledge Compilation Map. *J. Artif. Intell. Res.* 17 (2002), 229–264. doi:10.1613/jair.989

[31] Bruno De Finetti. 1929. Funzione caratteristica di un fenomeno aleatorio. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928*. 179–190.

[32] Bruno De Finetti. 1974. Theory of probability: a critical introductory treatment. (1974).

[33] Bruno De Finetti. 1980. On the condition of partial exchangeability. *Studies in inductive logic and probability* 2 (1980), 193–206.

[34] Guy Van den Broeck and Dan Suciu. 2017. Query Processing on Probabilistic Data: A Survey. *Found. Trends Databases* 7, 3-4 (2017), 197–341. doi:10.1561/1900000052

[35] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. 2011. Lifted Probabilistic Inference by First-Order Knowledge Compilation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, Toby Walsh (Ed.). IJCAI/AAAI, 2178–2185. doi:10.5591/978-1-57735-516-8/IJCAI11-363

[36] Persi Diaconis. 1988. Recent progress on de Finetti's notions of exchangeability. *Bayesian statistics* 3, 111-125 (1988), 13–14.

[37] Persi Diaconis and David Freedman. 1978. *De Finetti's generalizations of exchangeability*. Stanford University. Department of Statistics.

[38] Pedro M. Domingos, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. 2006. Unifying Logical and Statistical AI. 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,

2–9. http://www.aaai.org/Library/AAAI/2006/aaai06-001.php

[39] Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. 2015. ProbLog2: Probabilistic Logic Programming. 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, Vol. 9286)*, Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavaldà, Dino Pedreschi, Francesco Bonchi, Jaime S. Cardoso, and Myra Spiliopoulou (Eds.). Springer, 312–315. doi:10.1007/978-3-319-23461-8_37

[40] Florian Eggenberger and George Pólya. 1923. Über die statistik verketteter vorgänge. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 3, 4 (1923), 279–289.

[41] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Sht. Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. 2015. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory Pract. Log. Program.* 15, 3 (2015), 358–401. doi:10.1017/S1471068414000076

[42] Hal Finkel, David Poliakoff, Jean-Sylvain Camier, and David F Richards. 2019. Clangjit: Enhancing c++ with just-in-time compilation. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 82–95.

[43] Tor Erlend Fjelde, Kai Xu, David Widmann, Mohamed Tarek, Cameron Pfiffer, Martin Trapp, Seth D Axen, Xianda Sun, Markus Hauru, Penelope Yong, et al. 2025. Turing. jl: a general-purpose probabilistic programming language. *ACM Transactions on Probabilistic Machine Learning* (2025).

[44] Nicholas J. Foti, Jason Xu, Dillon Laird, and Emily B. Fox. 2014. Stochastic variational inference for hidden Markov models. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). 3599–3607. https://proceedings.neurips.cc/paper/2014/hash/865dfbde8a344b44095495f3591f7407-Abstract.html

[45] James Foulds, Levi Boyles, Christopher DuBois, Padhraic Smyth, and Max Welling. 2013. Stochastic Collapsed Variational Bayesian Inference for Latent Dirichlet Allocation. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery, New York, NY, USA, 446–454. doi:10.1145/2487575.2487697

[46] Apache Software Foundation. 2023. Apache Arrow. A cross-language development platform for in-memory analytics. https://arrow.apache.org. Accessed: 2023-04-01.

[47] Charles W. Fox and Stephen J. Roberts. 2012. A tutorial on variational Bayesian inference. *Artif. Intell. Rev.* 38, 2 (2012), 85–95. doi:10.1007/S10462-011-9236-8

[48] Norbert Fuhr. 1995. Probabilistic Datalog - A Logic For Powerful Retrieval Methods. In *SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, Washington, USA, July 9-13, 1995 (Special Issue of the SIGIR Forum)*, Edward A. Fox, Peter Ingwersen, and Raya Fidel (Eds.). ACM Press, 282–290. doi:10.1145/215206.215372

[49] Wolfgang Gatterbauer and Dan Suciu. 2014. Oblivious bounds on the probability of Boolean functions. *ACM Transactions on Database Systems (TODS)* 39, 1 (2014), 1–34.

[50] Wolfgang Gatterbauer and Dan Suciu. 2015. Approximate lifted inference with probabilistic databases. *Proc. VLDB Endow.* 8, 5 (2015), 629–640. doi:10.14778/2735479.2735494

[51] Renato Lui Geh and Denis Deratani Mauá. 2021. Learning probabilistic sentential decision diagrams under logic constraints by sampling and averaging. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2021, Virtual Event, 27-30 July 2021 (Proceedings of Machine Learning Research, Vol. 161)*, Cassio P. de Campos, Marloes H. Maathuis, and Erik Quaeghebeur (Eds.). AUAI Press, 2039–2049. https://proceedings.mlr.press/v161/geh21a.html

[52] Stuart Geman and Donald Geman. 1984. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans. Pattern Anal. Mach. Intell.* 6, 6 (1984), 721–741. doi:10.1109/TPAMI.1984.4767596

[53] 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

[54] Todd J. Green and Val Tannen. 2006. Models for Incomplete and Probabilistic Information. In *Current Trends in Database Technology - EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 4254)*, Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen (Eds.). Springer, 278–296. doi:10.1007/11896548_24

[55] Thomas L Griffiths and Mark Steyvers. 2004. Finding scientific topics. *Proceedings of the National academy of Sciences* 101, suppl 1 (2004), 5228–5235.

[56] Martin Grohe, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Peter Lindner. 2022. Generative Datalog with Continuous Distributions. *J. ACM* 69, 6 (2022), 46:1–46:52. doi:10.1145/3559102

[57] W Keith Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. (1970).

[58] Matthew D. Hoffman, David M. Blei, and Francis R. Bach. 2010. Online Learning for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta (Eds.). Curran Associates, Inc., 856–864. https://proceedings.neurips.cc/paper/2010/hash/71f6278d140af599e06ad9bf1ba03cb0-Abstract.html

[59] Matthew D. Hoffman, David M. Blei, Chong Wang, and John W. Paisley. 2013. Stochastic variational inference. *J. Mach. Learn. Res.* 14, 1 (2013), 1303–1347. doi:10.5555/2567709.2502622

[60] Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. 2020. Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 140:1–140:31. doi:10.1145/3428208

[61] Aaron Huber, Oliver Kennedy, Atri Rudra, Zhuoyue Zhao, Su Feng, and Boris Glavic. 2025. FastPDB: Towards Bag-Probabilistic Queries at Interactive Speeds. *Proc. ACM Manag. Data* 3, 1 (2025), 41:1–41:25. doi:10.1145/3709691

[62] Mathieu Huot, Matin Ghavami, Alexander K. Lew, Ulrich Schaechtle, Cameron E. Freer, Zane Shelby, Martin C. Rinard, Feras A. Saad, and Vikash K. Mansinghka. 2024. GenSQL: A Probabilistic Programming System for Querying Generative Models of Database Tables. *Proc. ACM Program. Lang.* 8, PLDI (2024), 790–815. doi:10.1145/3656409

[63] Tomasz Imielinski and Witold Lipski Jr. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (1984), 761–791. doi:10.1145/1634.1886

[64] Manfred Jaeger. 2004. Probabilistic Decision Graphs - Combining Verification And Ai Techniques For Probabilistic Inference. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 12, Supplement-1 (2004), 19–42. doi:10.1142/S0218488504002564

[65] 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. doi:10.1145/3422648.3422659

[66] Abhay Kumar Jha and Dan Suciu. 2011. Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams. In *Database Theory - ICDT 2011 (Lecture Notes in Computer Science, Vol. 6594)*. 121–132. doi:10.1007/978-3-642-20142-6_12

[67] Michael Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence Saul. 1999. An Introduction to Variational Methods for Graphical Models. *Machine Learning* 37 (01 1999), 183–233. doi:10.1023/A:1007665907178

[68] Angelika Kimmig, Bart Demoen, Luc De Raedt, Vítor Santos Costa, and Ricardo Rocha. 2011. On the implementation of the probabilistic logic programming language ProbLog. *Theory Pract. Log. Program.* 11, 2-3 (2011), 235–262. doi:10.1017/S1471068410000566

[69] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. 2014. Probabilistic Sentential Decision Diagrams. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*, Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8005

[70] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press. http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11886

[71] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2017. Automatic Differentiation Variational Inference. *J. Mach. Learn. Res.* 18 (2017), 14:1–14:45. http://jmlr.org/papers/v18/16-107.html

[72] Jean-Marie Lagniez and Pierre Marquis. 2017. An Improved Decision-DNNF Compiler.. In *IJCAI*, Vol. 17. 667–673.

[73] Yitao Liang, Jessa Bekker, and Guy Van den Broeck. 2017. Learning the Structure of Probabilistic Sentential Decision Diagrams. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*, Gal Elidan, Kristian Kersting, and Alexander Ihler (Eds.). AUAI Press. http://auai.org/uai2017/proceedings/papers/291.pdf

[74] Ove Lundberg. 1940. *On random processes and their application to sickness and accident statistics*. Ph. D. Dissertation. Almqvist & Wiksell.

[75] Hosam Mahmoud. 2008. *Pólya urn models*. Chapman and Hall/CRC.

[76] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. DeepProbLog: neural probabilistic logic programming. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 3753–3763.

[77] Vikash Mansinghka, Richard Tibbetts, Jay Baxter, Patrick Shafto, and Baxter Eaves. 2015. BayesDB: A probabilistic programming system for querying the probable implications of data. *CoRR* abs/1512.05006 (2015). arXiv:1512.05006 http://arxiv.org/abs/1512.05006

[78] Albert W Marshall and Ingram Olkin. 1990. Bivariate distributions generated from Pólya-Eggenberger urn models. *Journal of multivariate analysis* 35, 1 (1990), 48–65.

[79] Pedro Zuidberg Dos Martires, Luc De Raedt, and Angelika Kimmig. 2024. Declarative probabilistic logic programming in discrete-continuous domains. *Artif. Intell.* 337 (2024), 104227. doi:10.1016/J.ARTINT.2024.104227

[80] Lilith Mattei, Alessandro Antonucci, Denis Deratani Mauá, Alessandro Facchini, and Julissa Villanueva Llerena. 2020. Tractable inference in credal sentential decision diagrams. *Int. J. Approx. Reason.* 125 (2020), 26–48. doi:10.1016/J.IJAR.2020.06.005

[81] 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

[82] Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit. *http://mallet. cs. umass. edu* (2002).

[83] Wannes Meert. 2018. PySDD. In *Recent Trends in Knowledge Compilation, Report from Dagstuhl Seminar 17381*, Adnan Darwiche, Pierre Marquis, Dan Suciu, and Stefan Szeider (Eds.). Dagstuhl Reports, Vol. 7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 62–85. doi:10.4230/DagRep.7.9.62

[84] Niccolò Meneghetti and Ouael Ben Amara. 2022. Gamma Probabilistic Databases: Learning from Exchangeable Query-Answers. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*, Julia Stoyanovich, Jens Teubner, Paolo Guagliardo, Milos Nikolic, Andreas Pieris, Jan Mühlig, Fatma Özcan, Sebastian Schelter, H. V. Jagadish, and Meihui Zhang (Eds.). OpenProceedings.org, 2:260–2:273. doi:10.48786/EDBT.2022.14

[85] 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

[86] Cameron Moy, Jack Czenszak, John M. Li, Brianna Marshall, and Steven Holtzen. 2025. Roulette: A Language for Expressive, Exact, and Efficient Discrete Probabilistic Programming. *Proc. ACM Program. Lang.* 9, PLDI (2025), 2081–2105. doi:10.1145/3729334

[87] Christian Muise, Sheila A McIlraith, J Christopher Beck, and Eric I Hsu. 2012. Dsharp: Fast d-DNNF compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*. Springer, 356–361.

[88] Laura Isabel Galindez Olascoaga, Wannes Meert, Nimish Shah, Guy Van den Broeck, and Marian Verhelst. 2020. Discriminative Bias for Learning Probabilistic Sentential Decision Diagrams. In *Advances in Intelligent Data Analysis XVIII - 18th International Symposium on Intelligent Data Analysis, IDA 2020, Konstanz, Germany, April 27-29, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12080)*, Michael R. Berthold, Ad Feelders, and Georg Krempl (Eds.). Springer, 184–196. doi:10.1007/978-3-030-44584-3_15

[89] Hoifung Poon and Pedro M. Domingos. 2011. Sum-Product Networks: A New Deep Architecture. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, Fábio Gagliardi Cozman and Avi Pfeffer (Eds.). AUAI Press, 337–346. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2194&proceeding_id=27

[90] Luc de Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. 2020. From Statistical Relational to Neuro-Symbolic Artificial Intelligence. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization, 4943–4950. doi:10.24963/ijcai.2020/688 Survey track.

[91] Luc De Raedt and Kristian Kersting. 2008. Probabilistic Inductive Logic Programming. In *Probabilistic Inductive Logic Programming - Theory and Applications*, Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen H. Muggleton (Eds.). Lecture Notes in Computer Science, Vol. 4911. Springer, 1–27. doi:10.1007/978-3-540-78652-8_1

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

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

[94] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.

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

[96] Taisuke Sato. 1995. A Statistical Learning Method for Logic Programs with Distribution Semantics. In *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, Leon Sterling (Ed.). MIT Press, 715–729.

[97] Taisuke Sato and Yoshitaka Kameya. 1997. PRISM: A Language for Symbolic-Statistical Modeling. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*. Morgan Kaufmann, 1330–1339. http://ijcai.org/Proceedings/97-2/Papers/078.pdf

[98] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. ProvSQL: Provenance and Probability Management in PostgreSQL. *Proc. VLDB Endow.* 11, 12 (2018), 2034–2037. doi:10.14778/3229863.3236253

[99] Amir Shaikhha, Mohammad Dashti, and Christoph Koch. 2018. Push versus pull-based loop fusion in query engines. *J. Funct. Program.* 28 (2018), e10. doi:10.1017/S0956796818000102

[100] Claude E. Shannon. 1949. The synthesis of two-terminal switching circuits. *Bell Syst. Tech. J.* 28, 1 (1949), 59–98. doi:10.1002/J.1538-7305.1949.TB03624.X

[101] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. 2019. GANAK: A Scalable Probabilistic Exact Model Counter. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, 1169–1176. doi:10.24963/IJCAI.2019/163

[102] Yujia Shen, Arthur Choi, and Adnan Darwiche. 2016. Tractable Operations for Arithmetic Circuits of Probabilistic Models. 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.). 3936–3944. https://proceedings.neurips.cc/paper/2016/hash/5a7f963e5e0504740c3a6b10bb6d4fa5-Abstract.html

[103] Yujia Shen, Arthur Choi, and Adnan Darwiche. 2018. Conditional PSDDs: Modeling and Learning With Modular Knowledge. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 6433–6442. doi:10.1609/AAAI.V32I1.12119

[104] Yujia Shen, Anchal Goyanka, Adnan Darwiche, and Arthur Choi. 2019. Structured Bayesian Networks: From Inference to Learning with Routes. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 7957–7965. doi:10.1609/AAAI.V33I01.33017957

[105] Andy Shih and Stefano Ermon. 2020. Probabilistic circuits for variational inference in discrete graphical models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 389, 12 pages.

[106] 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).

[107] Sam Stites, John M. Li, and Steven Holtzen. 2025. Multi-Language Probabilistic Programming. *Proc. ACM Program. Lang.* 9, OOPSLA1 (2025), 1239–1266. doi:10.1145/3720482

[108] Sam Stites, Heiko Zimmermann, Hao Wu, Eli Sennesh, and Jan-Willem van de Meent. 2021. Learning proposals for probabilistic programs with inference combinators. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (Proceedings of Machine Learning Research, Vol. 161)*, Cassio de Campos and Marloes H. Maathuis (Eds.). PMLR, 1056–1066. https://proceedings.mlr.press/v161/stites21a.html

[109] Yee Whye Teh, David Newman, and Max Welling. 2006. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, Bernhard Schölkopf, John C. Platt, and Thomas Hofmann (Eds.). MIT Press, 1353–1360. https://proceedings.neurips.cc/paper/2006/hash/532b7cbe070a3579f424988a040752f2-Abstract.html

[110] 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, Vol. 9286)*, Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavaldà, Dino Pedreschi, Francesco Bonchi, Jaime S. Cardoso, and Myra Spiliopoulou (Eds.). Springer, 308–311. doi:10.1007/978-3-319-23461-8_36

[111] Pietro Totis, Jesse Davis, Luc De Raedt, and Angelika Kimmig. 2023. Lifted Reasoning for Combinatorial Counting. *J. Artif. Intell. Res.* 76 (2023), 1–58. doi:10.1613/JAIR.1.14062

[112] 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

[113] Efthymia Tsamoura, Víctor Gutiérrez-Basulto, and Angelika Kimmig. 2020. Beyond the Grounding Bottleneck: Datalog Techniques for Inference in Probabilistic Logic Programs. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February*

7-12, 2020. AAAI Press, 10284–10291. https://ojs.aaai.org/index.php/AAAI/article/view/6591

[114] Calin Rares Turliuc, Luke Dickens, Alessandra Russo, and Krysia Broda. 2016. Probabilistic abductive logic programming using Dirichlet priors. *International Journal of Approximate Reasoning* 78 (2016), 223–240.

[115] UCLA Automated Reasoning Group. 2015. ACE: A package for compiling Bayesian networks into arithmetic circuits. http://reasoning.cs.ucla.edu/ace/. Accessed: 2025-05-06.

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

[117] Hanna M Wallach, Iain Murray, Ruslan Salakhutdinov, and David Mimno. 2009. Evaluation methods for topic models. In *Proceedings of the 26th annual international conference on machine learning*. 1105–1112.

Received October 2025; revised January 2026; accepted February 2026