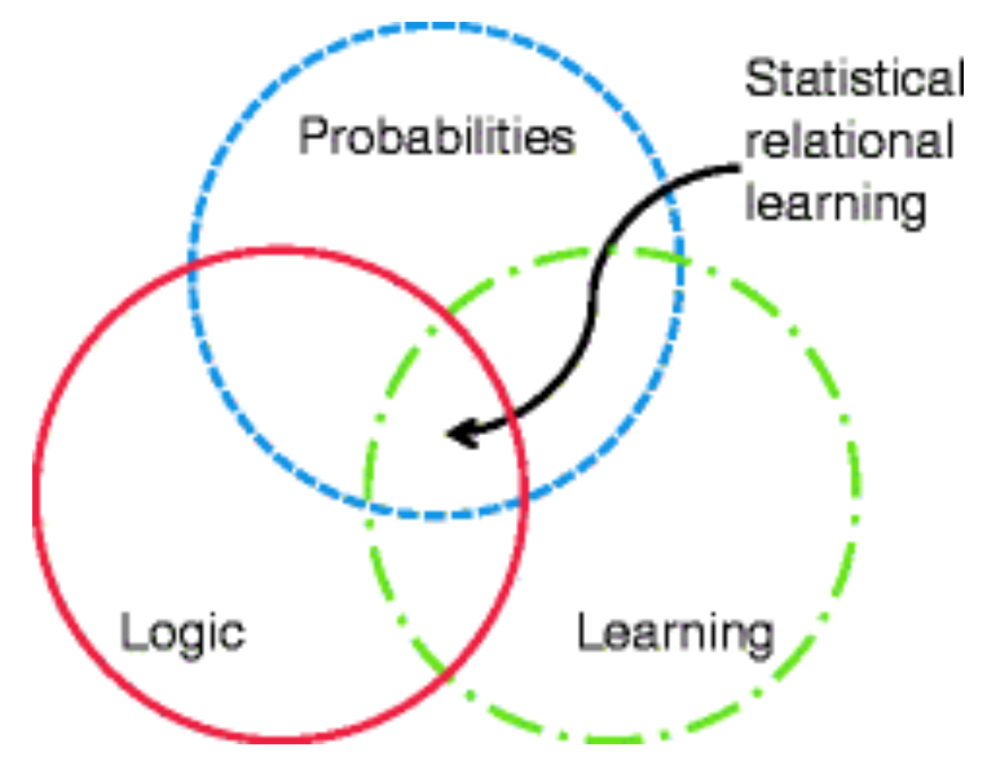
StarfishDB: A Query Execution Engine For Relational Probabilistic Programming

Ouael Ben Amara*, Sami Hadouaj*, Niccolò Meneghetti

Motivation & Context

Probabilistic programming has proven to be a highly successful framework for creating and implementing Bayesian statistical models. It simplifies the process for statisticians to express their probabilistic models and conduct inference without delving into the intricacies of their implementation.

Recently, a new breed of PP frameworks has been developed in the context of Statistical Relational Learning. This approach consists of a mix of logic programming, probabilistic inference, and learning. By employing logic for representing probabilistic programs, these frameworks offer enhanced statistical learning capabilities and improved performance.



StarfishDB is a probabilistic programming framework that is classified within the scope of relational statistical learning. It uses a restricted version of Probabilistic Programming Datalog[1] to express probabilistic programs, then employs advanced compilation tools to automatically produce an effective inference algorithm customized for the specified model.

Main Concepts

Probabilistic Program

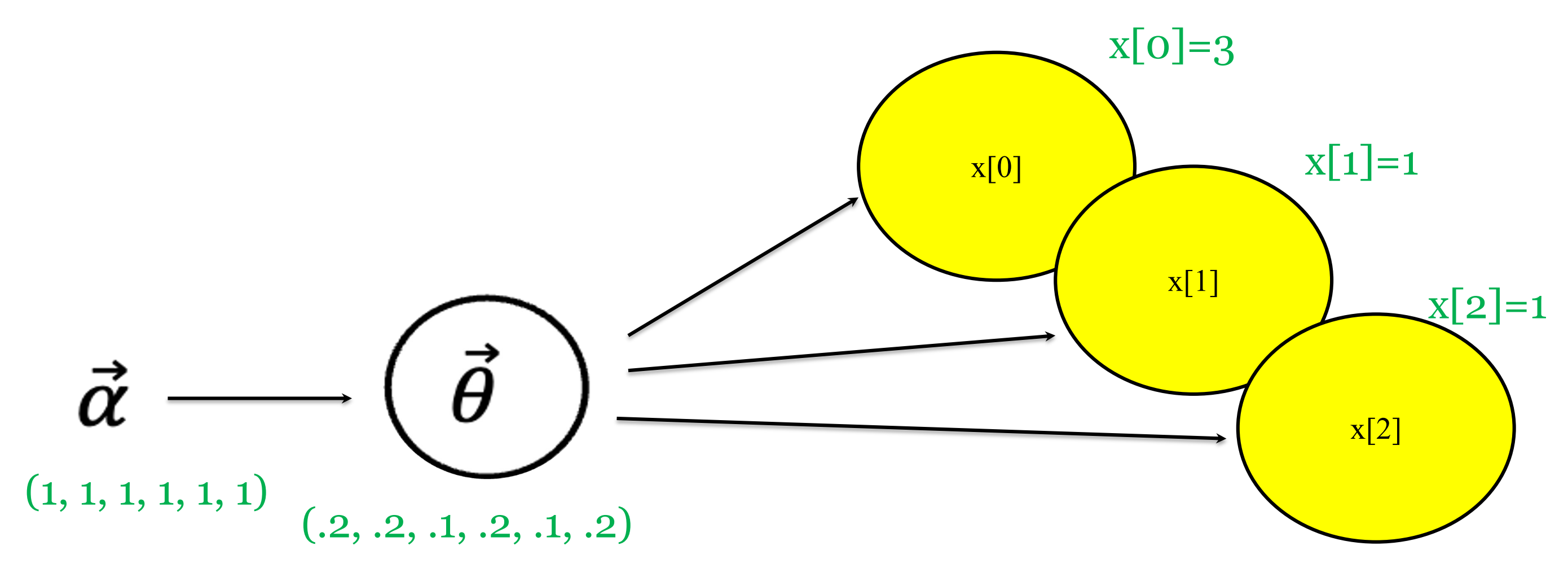
A probabilistic program is composed of a *stochastic generative process* and a set of *conditioning observations.*

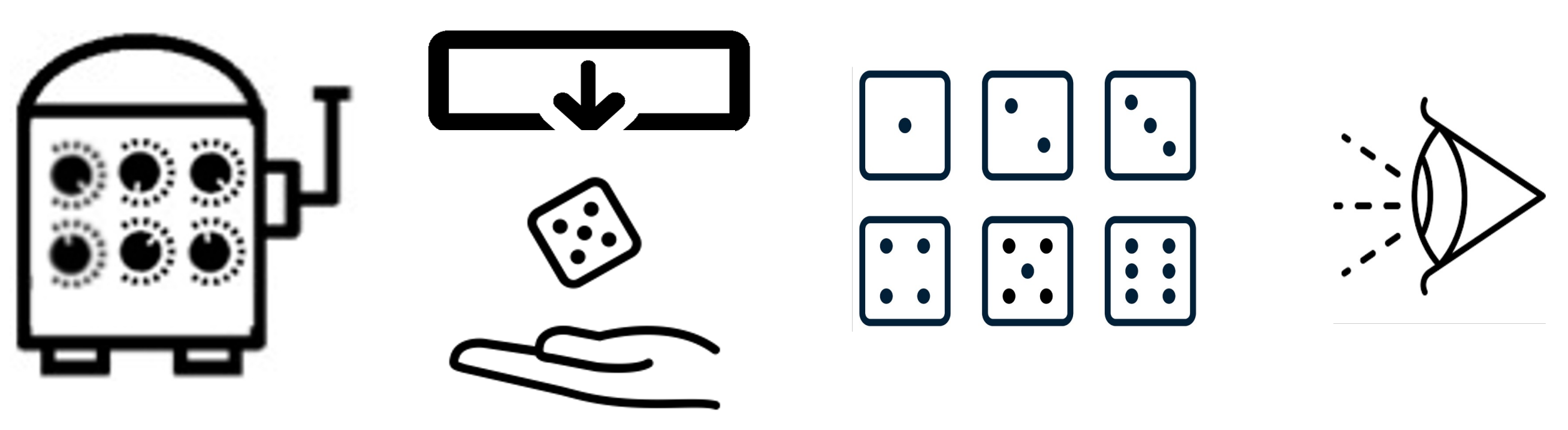
Exchangeability

We adopt the notion of exchangeability defined by Bruno Definetti. It entails that a set of variables are exchangeable if their joint distribution is invariant under any permutation.

Polya Die

A polya die is a simple Bayesian model composed of a categorical distribution paired with a Dirichlet prior. It consists of drawing a vector of probabilities theta from a Dirichlet distribution and using it to parametrize a categorical distribution





Probabilistic Datalog

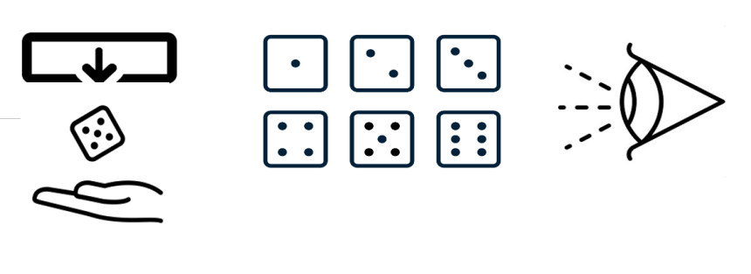
We use a restricted version of Probabilistic programming with Datalog[1] to represent probabilistic programs. This is done by defining two main rules that describe our Bayesian pipeline.

Rule1: Defines the instantiation of a die given a domain and a set of hyperparameters

$$\mathrm{lp}(\underline{VarId}, D, p \in \mathcal{S}_{|D|} \sim \mathcal{D}ir[\![H]\!]) \leftarrow \mathrm{dt}(VarId, D, H)$$

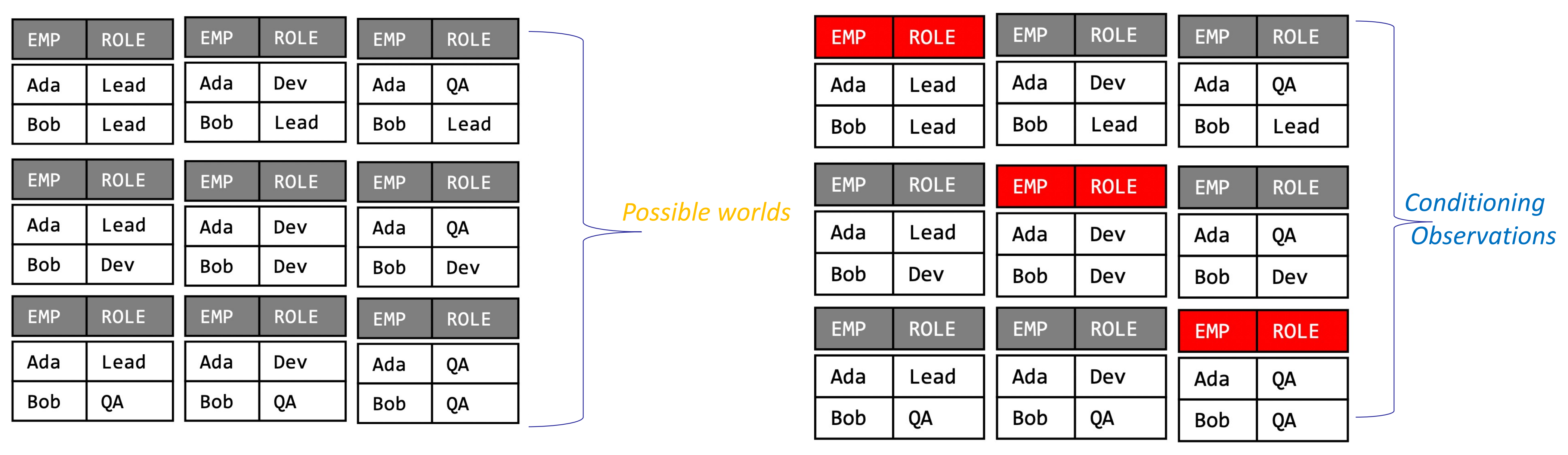
Rule2: characterizes the operation of throwing a die and denoting the observation. This is the sampling process

$$\mathrm{obs}(\underline{VarId, ObsId}, v \in D \sim Cat[\![P]\!]) \leftarrow \mathrm{lp}(VarId, D, P),\ \mathrm{sample}(VarId, ObsId)$$

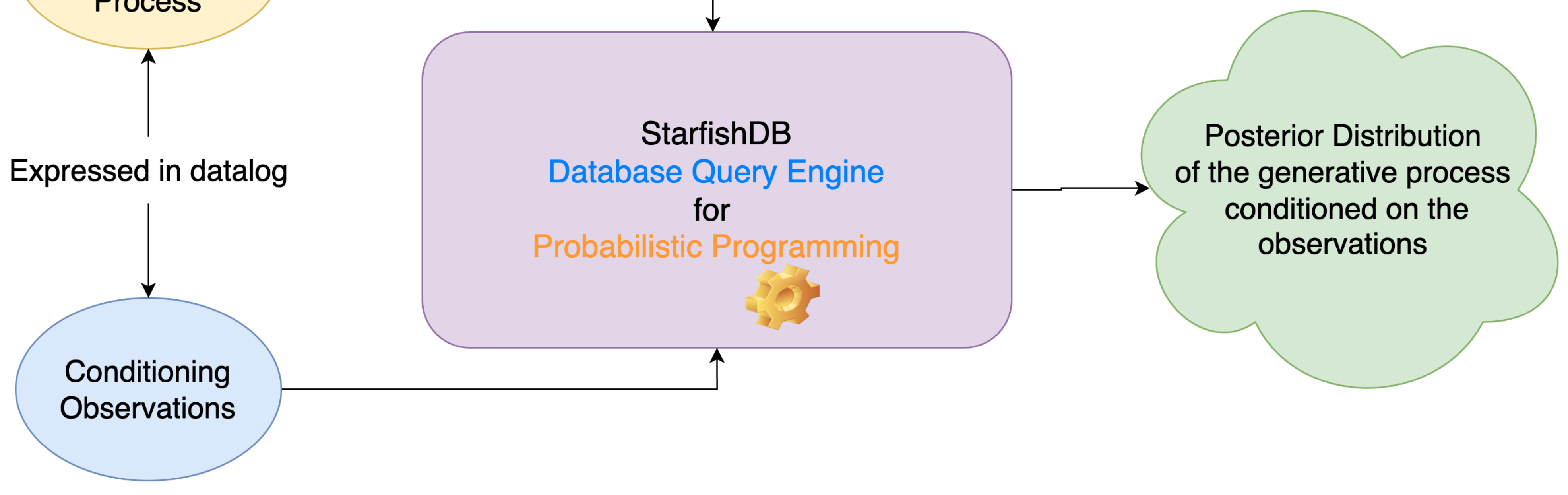


Relational Probabilistic Programming

A generative process represents the way all the possible worlds are generated. This is encoded using a set of polya dice. In the example below, we consider modeling the potential roles of two individuals, Ada and Bob, within a company. These roles could include lead, developer, or quality engineer. This can be represented using a Polya die with 3 faces corresponding to each role.
The conditioning observations are represented using query-answers. They are a set of constraints that represent the data used to condition the set of all possible words. C1 is an example of such constraints. In real-world PP, we will have several exchangeable constraints. This entails that they can be satisfied simultaneously since each constraint has its own die-rolls.



C1 : Ada and Bob have the same role


Stochastic Generative Process
Expressed in datalog
Conditioning Observations
StarfishDB
Database Query Engine
for
Probabilistic Programming
Posterior Distribution of the generative process conditioned on the observations

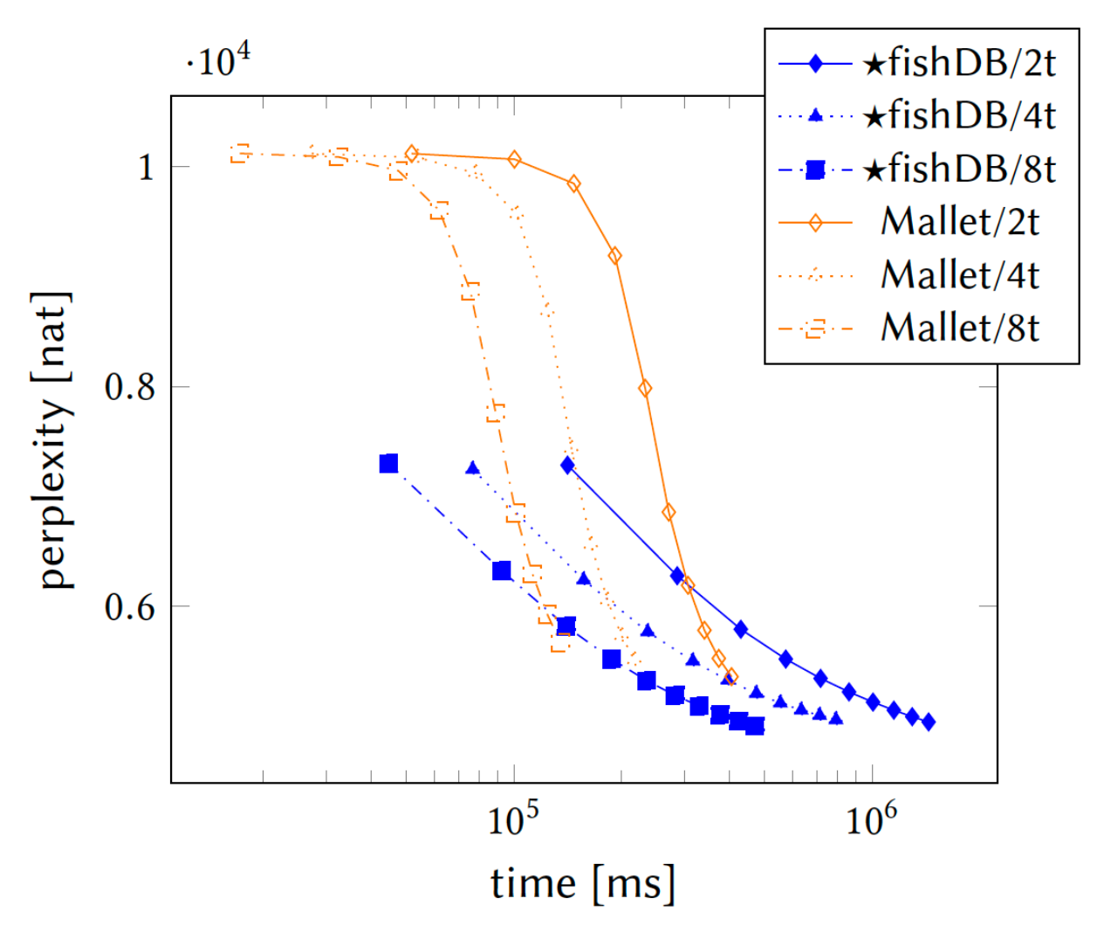
LDA with Probabilistic Datalog

$$\mathrm{dt}([\mathrm{red}, D], \mathrm{ts}, [1, 1, .., 1]) \leftarrow \mathrm{d}(D, P, W).$$
$$\mathrm{dt}([\mathrm{blue}, T], \mathrm{ws}, [1, 1, .., 1]) \leftarrow \mathrm{t}(T).$$
$$\mathrm{sample}([\mathrm{red}, D], P) \leftarrow \mathrm{d}(D, P, W).$$
$$\mathrm{sample}([\mathrm{blue}, T], [D, P]) \leftarrow \mathrm{d}(D, P, W), \mathrm{obs}([\mathrm{red}, D], P, T).$$
$$\mathrm{qa}^*(D, P, W) \leftarrow \mathrm{d}(D, P, W), \mathrm{obs}([\mathrm{blue}, T], [D, P], W).$$

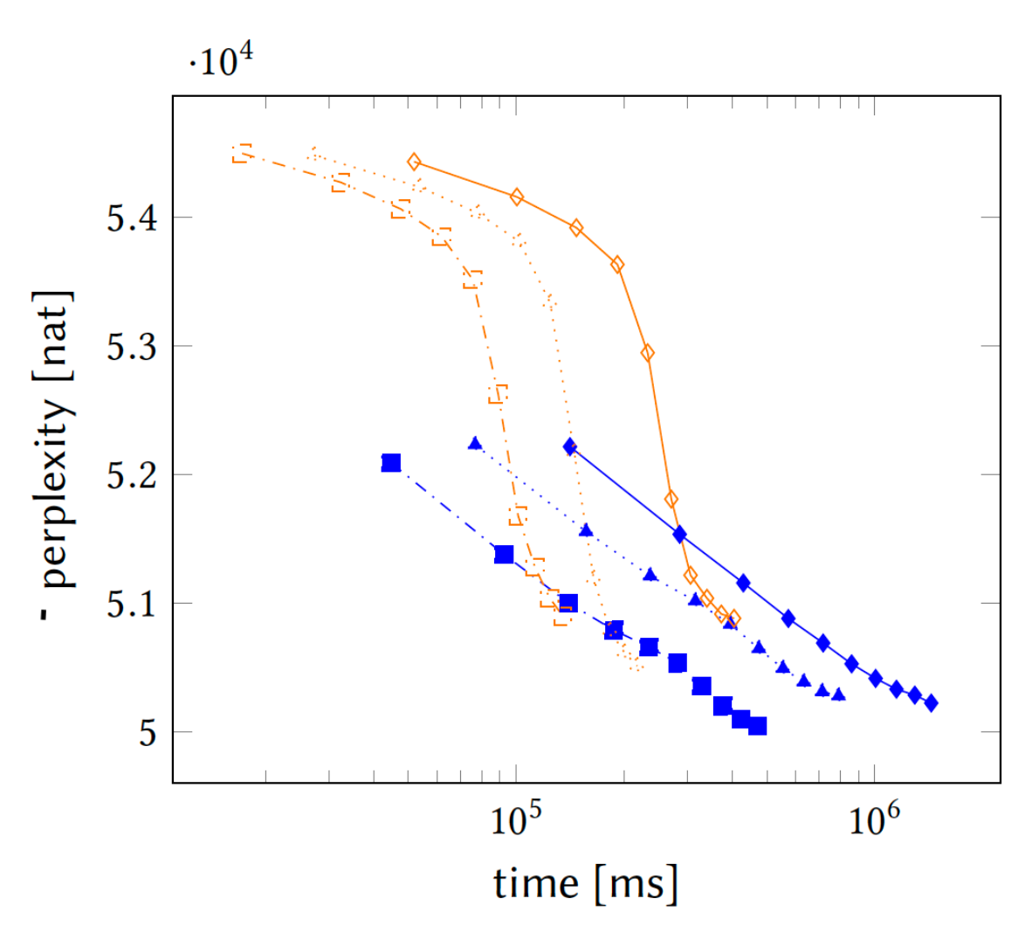
Just in Time Compilation

Our prototype consists of a query-exec-engine that extends the one provided by Apache Arrow (Acero). We start with a probabilistic program and we use relational operators in ACERO to ground it into a large collection of propositional constraints.
All these constraints will share the same factorization as they were generated by the same structure. This means that we can use JIT-compilation[2] to build a function that is just perfect for sampling a solution for these constraints, that has this specific structure.
This allows us to compile this function once and execute it many many times, over all the constraints.

Results



(a) Train-Set NYTIMES



(b) Test-Set NYTIMES

LDA Multithreaded 100 topics benchmarking on NYTIMES using 2, 4 and 8 threads

Conclusion

StarfishDB is a novel probabilistic programming framework that supports recursion and advanced factorization techniques to encode a wider range of probabilistic programs. It leverages just-in-time compilation to speed up inference and capitalize on the recurring patterns of the ground constraints. Our experiments show that there is no performance penalty for using our generic framework compared to other specialized tools.

References

[1] Bárány, Vince, Balder Ten Cate, Benny Kimelfeld, Dan Olteanu, and Zografoula Vagena. "Declarative probabilistic programming with datalog." ACM Transactions on Database Systems (TODS) 42, no. 4 (2017): 1-35

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