PPDL: Probabilistic Programming with Datalog

Balder ten Cate[1], Benny Kimelfeld[⋆2,1], and Dan Olteanu[3,1]

[1] LogicBlox, Inc., USA [2] Technion, Israel [3] University of Oxford, UK

1 Introduction

There has been a substantial recent focus on the concept of *probabilistic programming* [6] towards its positioning as a prominent paradigm for advancing and facilitating the development of machine-learning applications.[4] A probabilistic-programming language typically consists of two components: a specification of a stochastic process (the prior), and a specification of observations that restrict the probability space to a conditional subspace (the posterior). This paper gives a brief overview of *Probabilistic Programming DataLog* (PPDL), a recently proposed declarative framework for specifying statistical models on top of a database, through an appropriate extension of Datalog [1]. By virtue of extending Datalog, PPDL offers a natural integration with the database, and has a robust declarative semantics, that is, semantic independence from the algorithmic evaluation of rules, and semantic invariance under logical program transformations. It provides convenient mechanisms to allow common numerical probability functions as first-class citizens in the language; in particular, conclusions of rules may contain values drawn from such functions.

2 PPDL

The semantics of a PPDL program is a probability distribution over the possible outcomes of the input database with respect to the program. These outcomes are minimal solutions with respect to a related program that involves existentially quantified variables in conclusions. Observations are incorporated by means of logical integrity constraints. As argued in [1], the ability to express probabilistic models *concisely* and *declaratively* in a Datalog extension, with probability distributions as first-class citizens, is what sets PPDL apart from the wealth of literature on probabilistic Datalog [3], probabilistic databases [8,11], and Markov Logic Networks [4, 7, 10]. In the remaining of this section we introduce PPDL using an example program, and show how to interpret it probabilistically.

Our example is inspired by the burglar example of Pearl that has been frequently used for illustrating probabilistic programming. This example models a statistical process of alarming due to burglaries and earthquakes, and the goal is

⋆ Taub Fellow – supported by the Taub Foundation

[4] An effort in this direction is led by DARPA's *Probabilistic Programming for Advancing Machine Learning* (PPAML) program.

House	
id	*city*
NP1	Napa
NP2	Napa
YC1	Yucaipa

Business	
id	*city*
NP3	Napa
YC1	Yucaipa

City	
name	*burglaryrate*
Napa	0.03
Yucaipa	0.01

ObservedAlarm
unit
NP1
YC1
YC2

Fig. 1. Database instance in the running example.

to estimate the likelihood that a given collection of alarms indicates these alarming events. Consider a database consisting of the following relations: House(h, c) represents houses h and their location cities c, Business(b, c) represents businesses b and their location cities c, City(c, r) represents cities c and their associated burglary rates r, and ObservedAlarm(x) represents units (houses or businesses) x where the alarm went off. These are the *EDB* relations that are not changed by the program. Figure 1 shows an instance over the schema. Now consider the PPDL program $\mathcal{P}$ in Figure 2, where some rules use the Flip distribution in their heads. The first rule states, intuitively, that for every fact of the form City(c, r), there must be a fact Earthquake(c, y) where y is drawn from the Flip (Bernoulli) distribution with the parameter 0.01. The fourth rule states that a burglary happens in a unit (house or business) with probability r, where r is a number that represents the rate of burglaries in the city of the unit (note that we represent by Burglary$(x, c, 1)$ and Burglary$(x, c, 0)$ the fact that a Burglary *did*, respectively, *did not* happen at unit x in city c; likewise for Earthquake and Trig). Finally, c1 is a constraint stating that Alarm and ObservedAlarm have the same tuples.

We now address the semantics of a program. What does it mean for a rule head like Earthquake$(c, \mathsf{Flip}[0.01])$ to be *satisfied*? What if this fact is derived by multiple, or even equivalent, rules? Do we need to sample more than once? The probabilistic semantics of the above PPDL program is established via an extension to Datalog, named Datalog$^{\exists}$, where rule heads can have existential quantifiers. Datalog$^{\exists}$ rules (a.k.a. *existential rules*, which are syntactically isomorphic to tuple-generating dependencies), have been used extensively in many areas, including data exchange [5] and ontological reasoning [2, 9]. Our PPDL

1. Earthquake$(c, \mathsf{Flip}[0.01])$ $\leftarrow$ City(c, r)
2. Unit(h, c) $\leftarrow$ Home(h, c)
3. Unit(b, c) $\leftarrow$ Business(b, c)
4. Burglary$(x, c, \mathsf{Flip}[r])$ $\leftarrow$ Unit(x, c) , City(c, r)
5. Trig$(x, \mathsf{Flip}[0.6])$ $\leftarrow$ Unit(x, c) , Earthquake$(c, 1)$
6. Trig$(x, \mathsf{Flip}[0.9])$ $\leftarrow$ Burglary$(x, c, 1)$
7. Alarm(x) $\leftarrow$ Trig$(x, 1)$

c1. Alarm(x) $\leftrightarrow$ ObservedAlarm(x)

Fig. 2. PPDL program $\mathcal{P}$ for Pearl's burglar example

1. $\exists y\ \text{Earthquake}_2^{\mathsf{Flip}}(c, y, 0.01) \leftarrow \text{City}(c, r)$
2. $\text{Unit}(h, c) \leftarrow \text{Home}(h, c)$
3. $\text{Unit}(b, c) \leftarrow \text{Business}(b, c)$
4. $\exists y\ \text{Burglary}_3^{\mathsf{Flip}}(x, c, y, r) \leftarrow \text{Unit}(x, c)\,,\ \text{City}(c, r)$
5. $\exists y\ \text{Trig}_2^{\mathsf{Flip}}(x, y, 0.6) \leftarrow \text{Unit}(x, c)\,,\ \text{Earthquake}(c, 1)$
6. $\exists y\ \text{Trig}_2^{\mathsf{Flip}}(x, y, 0.9) \leftarrow \text{Burglary}(x, c, 1)$
7. $\text{Alarm}(x) \leftarrow \text{Trig}(x, 1)$
8. $\text{Earthquake}(c, d) \leftarrow \text{Earthquake}_2^{\mathsf{Flip}}(c, d, p)$
9. $\text{Burglary}(x, c, b) \leftarrow \text{Burglary}_3^{\mathsf{Flip}}(x, c, b, p)$
10. $\text{Trig}(x, y) \leftarrow \text{Trig}_2^{\mathsf{Flip}}(x, y, p)$

Fig. 3. The Datalog$^{\exists}$ program $\widehat{\mathcal{P}}$ for the PPDL program $\mathcal{P}$ in Figure 2

program $\mathcal{P}$ gives rise to the Datalog$^{\exists}$ program $\widehat{\mathcal{P}}$ in Figure 3. Note that this program does not take into account the constraints. To illustrate the translation, note how rule 6 in $\mathcal{P}$ becomes rule 6 in $\widehat{\mathcal{P}}$. A special, *distributional* relation symbol $\text{Trig}_2^{\mathsf{Flip}}$ is created for Trig that captures the intention of the rule: whenever the premise holds (there is a Burglary at a unit x), then there exists a fact $\text{Trig}_2^{\mathsf{Flip}}(x, y, 0.9)$ where y is drawn from a Bernoulli distribution with parameter 0.9. Rule 10 is implicitly added to update Trig with the content of $\text{Trig}_2^{\mathsf{Flip}}$, where the additional parameter (i.e., the above 0.9) is projected out.

In the absence of constraints, a possible outcome of an input database instance I (a "possible world") is a minimal super-instance of I that satisfies the rules of the program. One possible outcome of the input instance in Figure 1 with respect to the rules of $\mathcal{P}$ is the database instance formed by the input instance and the relations in Figure 4. Each tuple of a distributional relation has a *weight*, which is the probability of the random choice made for that fact. For presentation's sake, the sampled values are under the attribute name *draw*. Ignoring the constraints (c1 in the example), the probability of this outcome is the product of all of the numbers in the columns titled "$w(f)$," that is, $0.01 \times 0.99 \times 0.03 \times \cdots \times 0.4$. Note that this multiplication is an instance of the *chain rule* $\Pr(A_1 \wedge \cdots \wedge A_n) = \Pr(A_1) \times \Pr(A_2|A_1) \times \ldots$, and does not reflect an assumption of independence among the involved draws. One needs to show that this formula gives a proper probability space (i.e., the probabilities of all possible worlds sum up to 1). We do so via an adaptation of the *chase* procedure [1].

Constraints, such as c1 in our example, do not trigger generation of tuples, but rather have the semantics of conditional probability: violating possible worlds (where Alarm is different from ObservedAlarm) are eliminated, and the probability is normalized across the remaining worlds. Hence, we unify the concept of *observations* in Bayesian statistics with that of *integrity constraints* in databases.

In summary, a PPDL program associates to every given input instance a probability distribution over possible outcomes. One can then, for example, ask for the marginal probability of an event such as $\text{Burglary}(\mathsf{NP1})$. Standard techniques

Earthquake$_2^{\mathsf{Flip}}$

city	*draw*	param	$w(f)$
Napa	1	0.01	0.01
Yucaipa	0	0.01	0.99

Earthquake

city	*draw*
Napa	1
Yucaipa	0

Burglary$_3^{\mathsf{Flip}}$

unit	*city*	*draw*	param	$w(f)$
NP1	Napa	1	0.03	0.03
NP2	Napa	0	0.03	0.97
NP3	Napa	1	0.03	0.03
YU1	Yucaipa	0	0.01	0.99

Alarm

unit
NP1
NP2

Burglary

unit	*city*	*draw*
NP1	Napa	1
NP2	Napa	0
NP3	Napa	1
YU1	Yucaipa	0

Unit

id	*city*
NP1	Napa
NP2	Napa
NP3	Napa
YU1	Yucaipa

Trig$_2^{\mathsf{Flip}}$

unit	*draw*	param	$w(f)$
NP1	1	0.9	0.9
NP3	0	0.9	0.1
NP1	1	0.6	0.6
NP2	1	0.6	0.6
NP3	0	0.6	0.4

Trig

unit	*draw*
NP1	1
NP2	1
NP3	0

Fig. 4. An outcome of the instance in Figure 1 with respect to the PPDL program $\mathcal{P}$.

from the probabilistic programming literature (analytical, as lifted inference, or sampling-based, as MCMC) can be used to answer such questions.

3 Discussion

Currently, PPDL semantics supports only discrete numerical distributions (e.g., Poisson). But even then, the space of possible outcomes may be uncountable (as possible outcomes may be infinite). We have defined a probability measure over possible outcomes by applying the known concept of *cylinder sets* to a probabilistic chase procedure. We have also shown that the resulting semantics is robust under different chases; moreover, we have identified conditions guaranteeing that all possible outcomes are finite (and then the probability space is discrete) [1]. The framework has a natural extension to continuous distributions (e.g., Gaussian or Pareto), though this requires a nontrivial generalization of our semantics. Additional future directions include an investigation of semantic aspects of expressive power, tractability of inference, and a practical implementation (e.g., corresponding sampling techniques).

Acknowledgements

We are thankful to Molham Aref, Vince Bárány, Todd J. Green and Emir Pasalic Zografoula Vagena for insightful discussions and feedback on this work. We are grateful to Kathleen Fisher and Suresh Jagannathan for including us in DARPA's PPAML initiative; this work came from our efforts to design translations of probabilistic programs into statistical solvers.

References

1. V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena. Declarative statistical modeling with Datalog. *CoRR*, abs/1412.2221, 2014.
2. A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010.
3. D. Deutch, C. Koch, and T. Milo. On probabilistic fixpoint and Markov chain query languages. In *PODS*, pages 215–226, 2010.
4. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on AI and Machine Learning. Morgan & Claypool Publishers, 2009.
5. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, volume 2572 of *LNCS*, pages 207–224. Springer, 2003.
6. N. D. Goodman. The principles and practice of probabilistic programming. In *POPL*, pages 399–402, 2013.
7. G. Gottlob, T. Lukasiewicz, M. Martinez, and G. Simari. Query answering under probabilistic uncertainty in Datalog+/ ontologies. *Annals of Math.& AI*, 69(1):37–72, 2013.
8. B. Kimelfeld and P. Senellart. Probabilistic XML: models and complexity. In *Adv. in Probabl. Databases for Uncertain Information Management*, volume 304 of *Studies in Fuzziness and Soft Computing*, pages 39–66. 2013.
9. M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI*, pages 963–968, 2011.
10. F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in Markov Logic Networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.
11. D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.