# Probabilistic Logic Programs: Unifying Program Trace and Possible World Semantics

Angelika Kimmig and Luc De Raedt
KU Leuven
{firstname.lastname}@cs.kuleuven.be

Probabilistic graphical models (PGMs) have been popular in artificial intelligence and machine learning ever since they were introduced by Judea Pearl in the late 80s. Today, they enable numerous applications in domains ranging from robotics to natural language processing and bio-informatics. However, PGMs essentially define a joint probability distribution of a fixed and finite set of variables, and in this way, they are akin to a propositional logic. Since the early 90s, researchers have contributed many formalisms for making PGMs more expressive, and able to naturally deal with a variable number of objects as well as the relationships amongst them, just like first-order logic. Researchers gathered in a field that became known as *statistical relational learning* (SRL) [1], [2] and pursued the quest for the ultimate "universal" representation and accompanying inference and learning procedures. A good overview of this productive research period is given in [3]. Russell distinguishes two types of probabilistic representations, the first of which defines a "possible world" semantics and the second a semantics that is based on "probabilistic execution traces". The possible world semantics provides the underlying intuition for probabilistic logics such as Markov Logic [4], BLOG [5], probabilistic logic programming under Sato's distribution semantics [6], and probabilistic databases [7], the trace semantics for probabilistic programming (PP) languages such as IBAL [8], Church [9] and Anglican [10].

While both types of representations were created within the field of SRL and were intended to be used for the same types of applications, in the past 5 to 8 years SRL and PP have been studied almost in isolation and now have a quite different focus. In probabilistic programming [11], the focus is on "functional and imperative programs", on modeling continuous random variables, on (Markov Chain) Monte-Carlo techniques for probabilistic inference, and on a Bayesian machine learning perspective going back to the BUGS system [12]. The study of probabilistic programming is especially popular in the machine learning and programming language communities. In contrast, in statistical relational artificial intelligence [2], the focus is on logical and database representations, on discrete distributions and on knowledge compilation and "lifted inference" (reasoning at an abstract level – without having to ground out variables over domains), and on learning the structure of the model. These topics are popular within the artificial intelligence and database community.

We argue that probabilistic logic programming (PLP) [13], whose rich history goes back to the early 90s with results by David Poole [14] and Taisuke Sato [6], is uniquely positioned in that it naturally connects these two views into a single formalism with rigorously defined semantics.

More specifically, in this note we show how probabilistic logic programs possess not only a possible world semantics but also a program trace semantics, which they inherit from traditional logic programs. Furthermore, as extensions of Prolog, probabilistic logic programming languages are Turing equivalent, a property that they share with probabilistic programming languages extending traditional functional or imperative programming languages, and which distinguishes them from many of the statistical relational learning formalisms such as probabilistic databases [7], Markov Logic [4] or PSL [15].

## I. Probabilistic Logic Programs

We briefly summarize the distribution semantics in its most basic form, that is, for definite clause programs with a finite set of random variables expressed through probabilistic facts. We refer to [6] for the general definition in terms of a base distribution over a countably infinite set of random variables, to [16] for the extension to normal logic programs (i.e., with negation), and to [13] for an overview of programming concepts within this framework. A probabilistic logic program consists of a set $R$ of definite clauses or *rules* $\mathtt{h}$ :- $\mathtt{b_1}$, ... , $\mathtt{b_n}$, meaning that if all atoms $\mathtt{b_i}$ are true, the atom $\mathtt{h}$ has to be true as well, and a set $F$ of ground facts $f$, each of them labeled with a probability $p$, written as *probabilistic fact* $p :: f$. We use the following program with four probabilistic facts and two rules as a running example:

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
smokes(X) :- stress(X).
smokes(X) :- influences(Y,X), smokes(Y).
```

Each probabilistic fact corresponds to a Boolean random variable that is *true* with probability $p$ and *false* with probability $1-p$. Assuming that all these random variables are independent, we obtain the following probability distribution $P_F$ over truth value assignments to these random variables and their corresponding sets of ground facts $F' \subseteq F$:

$$P_F(F') = \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i) \qquad (1)$$

For instance, the probability of the truth value assignment that sets `stress(bob)` to false and the other three probabilistic facts to true is $0.8 \cdot (1-0.4) \cdot 0.6 \cdot 0.2 = 0.0576$. As each logic program obtained by choosing a truth value for every probabilistic fact has a unique least Herbrand model (i.e., a unique least model using only symbols from the program,

TABLE I. POSSIBLE WORLDS WHERE `smokes(carl)` IS TRUE

| `st(ann)` | `st(bob)` | `infl(ann,bob)` | `infl(bob,carl)` | $P_F(F')$ |
|---|---|---|---|---|
| true | false | true | true | 0.0576 |
| true | true | true | true | 0.0384 |
| true | true | false | true | 0.0256 |
| false | true | true | true | 0.0096 |
| false | true | false | true | 0.0064 |

cf. Section II-A), which we call a *possible world*, $P_F$ can be used to define the *success probability* $P(q)$ of a ground query $q$, that is, the probability that $q$ is true in a randomly chosen such program, as the sum over all programs that entail $q$:

$$P(q) := \sum_{\substack{F' \subseteq F \\ F' \cup R \models q}} P_F(F') \tag{2}$$

$$= \sum_{\substack{F' \subseteq F \\ F' \cup R \models q}} \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i)\,. \tag{3}$$

For the truth value assignment above, the least Herbrand model additionally sets `smokes(ann)`, `smokes(bob)` and `smokes(carl)` to true, and all unmentioned atoms to false. Table I lists the truth value assignments for the probabilistic facts in all possible worlds where query `smokes(carl)` is true, together with their probabilities.

Distributional clauses [17] generalize the semantics to also allow base variables with infinite domains (under mild validity conditions, cf. [17]), where the set of possible worlds is no longer denumerable. In this case, the base distribution uses comparison predicates to map random variables into probabilistic facts. More precisely, a *distributional clause* is a clause of the form $\mathtt{h} \sim \mathcal{D}$ :- $\mathtt{b_1}, \ldots, \mathtt{b_n}$. where $\sim$ is a binary predicate used in infix notation. The head $(\mathtt{h} \sim \mathcal{D})\theta$ of a distributional clause is defined for a grounding substitution $\theta$ whenever $(\mathtt{b_1}, \ldots, \mathtt{b_n})\theta$ is true in the semantics of the logic program. Then the distributional clause defines the random variable $\mathtt{h}\theta$ as being distributed according to the associated distribution $\mathcal{D}\theta$. Possible distributions include finite discrete distributions such as a uniform distribution, discrete distributions over infinitely many values, such as a Poisson distribution, and continuous distributions such as Gaussian or Gamma distributions. The outcome of a random variable $h$ is represented by the term $\simeq(h)$. Both random variables $h$ and their outcome $\simeq(h)$ can be used as other terms in the program. However, the typical use of terms $\simeq(h)$ is inside comparison predicates such as `equal/2` or `lessthan/2`, which realize an alternative definition of the probabilistic facts in the base distribution $P_F$ above. Indeed, depending on the value of $\simeq(h)$ (which is determined probabilistically) they will be true or false.

## II. THREE SEMANTICS

In logic programming, one distinguishes three equivalent ways of defining the semantics of a logic program (LP).

### A. Declarative semantics

The **declarative** semantics of an LP is based on the standard model-theoretic semantics of first-order logic, namely it is the intersection of all Herbrand models of the program, called the least Herbrand model. As discussed above, this is the basis for the distribution semantics, as the truth values of the probabilistic facts $F'$ in Equation (1) together with the definite clauses fully determine the resulting possible world, which corresponds to the unique least Herbrand model.

This view is in line with the predominant view in statistical relational learning which defines the semantics as a probability distribution over possible worlds, which are interpretations of a logic theory or extensions of a database schema. That is, a possible world assigns truth values to all the ground atoms, which then serve as random variables. Thus this view does not only hold for probabilistic logic programs [13], but also for probabilistic databases and for Markov Logic [4].

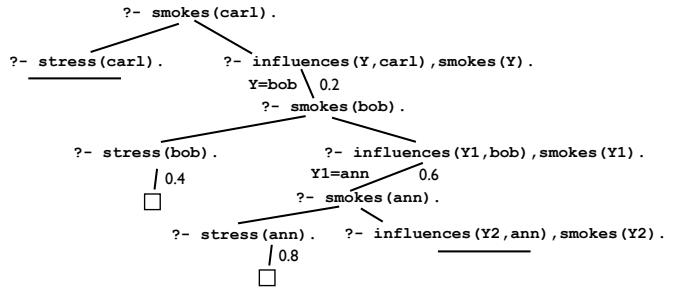


Fig. 1. Operational semantics: SLD tree for `smokes(carl)`, with two traces `influences(bob,carl),stress(bob)` and `influences(bob,carl),influences(ann,bob),stress(ann)` corresponding to the last four and the first two worlds in Table I, respectively.

### B. Operational semantics

The **operational** semantics of an LP is given by executions of the program, that is, refutation proofs of a query using resolution, also called *backward reasoning*. The SLD tree of a query and a program provides a visualization of the space of these executions, where the root of the tree corresponds to the query, other nodes correspond to subgoals reached during execution, and edges to resolution steps connecting their input and output goals. Proofs of the query correspond to branches ending in nodes labeled with the empty goal □. Figure 1 shows the SLD tree for our example query `smokes(carl)`, where edges corresponding to resolution steps with probabilistic facts are labeled with the corresponding probability. Note that as logic programs are inherently non-deterministic in nature, a query can have several proofs, and thus several program traces. In our example, the non-deterministic choice between the two rules leads to two traces for the query, and the sets of possible worlds admitting each trace are not disjoint, as they share the second world in Table I. The program trace semantics of PLP has to take into account the interaction between such non-deterministic choices and the probabilistic choices in the program. This is achieved by either ensuring that all choices are governed by probability distributions, e.g., [18], [19], in which case every possible world admits at most one trace, or through symbolic extensions of traces that restore this property in the presence of non-determinism, e.g., [16], [20]. In our example, the latter could be achieved by extending the second trace to also set `stress(bob)` to false, which restricts this trace to the first world in Table I.

This view is in line with the dominant view in probabilistic functional and imperative programming which defines a distribution over possible executions traces, followed by well-known languages such as IBAL [8], Church [9], Anglican [10], and Stan [21]. Adopting the operational semantics of PLP, it is possible to emulate the behavior of functional probabilistic programs. Indeed, it is well-known that a function $y = f(x_1, \ldots, x_n)$ can be interpreted as a relation $r(x_1, \ldots, x_n, y)$ over its inputs and result, and probabilistic functional programs can thus be mapped into probabilistic logic programs defining the same distribution over execution traces. One important difference between current PLP languages and functional languages such as Church is that the former generally either memoize all random variables, i.e., each mention of a random variable within a possible world or program trace refers to the same value, as in ProbLog [20] and ICL [16], or memoize no random variable, i.e., consider each mention as a fresh, independent random variable, as in PRISM [19], whereas functional languages allow a choice between the two for each random variable. However, this distinction only affects the set $F$ of probabilistic facts, which contains several independent copies of a fact if the PLP language does not use memoization, and a single one in case the language does use memoization. The semantics is well-defined in either case, but users need to be aware of the choice made by the system developers; we refer to [13, Sec. 4.4] for a detailed discussion.

### C. Denotational semantics

The **denotational** semantics of an LP associates with the program a function over the domain computed by the program, the so-called $T_P$-operator, and defines the meaning of the program as the least fixpoint of that function (if the fixpoint exists). For PLP, this *forward reasoning* provides a second natural interpretation of program traces as traces of least fixpoint computations [22]. Each possible world gives rise to one such trace, starting from the selected facts $F'$ in Equation (1). For instance, in the last possible world in Table I, the first step of the fixpoint computation sets `smokes(bob)` to true based on the first rule, and the next step adds `smokes(carl)` due to the second rule, after which no more atoms can be added. This corresponds to the generative model view in programming languages such as IBAL, BLOG and distributional clauses, which is also in line with the possible world view.

## III. Conclusion

The importance of these three different views on the semantics of PLP is that since they are known to coincide, inference for PLP can safely be based on techniques developed for any of these views. Indeed, while the backward view used by the operational semantics has been predominant for discrete PLPs, e.g. [14], [6], [20] and many others, more recent work has demonstrated that forward reasoning following the denotational semantics further increases scalability [23]. For distributional clauses, effective inference relies on combinations of both reasoning directions [17], [24].

*Acknowledgments* Angelika Kimmig is supported by the Research Foundation Flanders (FWO Vlaanderen).

## References

[1] L. Getoor and B. Taskar, Eds., *Statistical Relational Learning*. The MIT press, 2007.

[2] L. De Raedt, K. Kersting, S. Natarajan, and D. Poole, "Statistical relational artificial intelligence: Logic, probability and computation," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2016.

[3] S. J. Russell, "Unifying logic and probability," *Commun. ACM*, vol. 58, no. 7, pp. 88–97, 2015.

[4] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[5] B. Milch, B. Marthi, S. J. Russell, D. Sontag, D. L. Ong, and A. Kolobov, "BLOG: Probabilistic models with unknown objects," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005.

[6] T. Sato, "A statistical learning method for logic programs with distribution semantics," in *Proceedings of the 12th International Conference on Logic Programming (ICLP-95)*, 1995.

[7] D. Suciu, D. Olteanu, C. Ré, and C. Koch, *Probabilistic Databases*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[8] A. Pfeffer, "IBAL: A probabilistic rational programming language," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[9] N. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," in *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI-08)*, 2008.

[10] F. Wood, J. W. van de Meent, and V. Mansinghka, "A new approach to probabilistic programming inference," in *Proceedings of the International conference on Artificial Intelligence and Statistics*, 2014, pp. 1024–1032.

[11] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani, "Probabilistic programming," in *Int. Conference on Software Engineering (ICSE Future of Software Engineering)*. ACM, 2014, pp. 167–181.

[12] D. Spiegelhalter, A. Thomas, N. Best, and W. Gilks, "Bugs 0.5: Bayesian inference using gibbs sampling manual (version ii)," *MRC Biostatistics Unit, Institute of Public Health, Cambridge, UK*, pp. 1–59, 1996.

[13] L. De Raedt and A. Kimmig, "Probabilistic (logic) programming concepts," *Machine Learning*, vol. 100, no. 1, pp. 5–47, 2015.

[14] D. Poole, "Probabilistic Horn abduction and Bayesian networks," *Artificial Intelligence*, vol. 64, pp. 81–129, 1993.

[15] S. H. Bach, M. Broecheler, B. Huang, and L. Getoor, "Hinge-loss markov random fields and probabilistic soft logic," *ArXiv:1505.04406 [cs.LG]*, 2015.

[16] D. Poole, "Abducing through negation as failure: stable models within the independent choice logic," *Journal of Logic Programming*, vol. 44, no. 1-3, pp. 5–35, 2000.

[17] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt, "The magic of logical inference in probabilistic programming," *Theory and Practice of Logic Programming*, vol. 11, no. (4–5), pp. 663–680, 2011.

[18] S. Muggleton, "Stochastic logic programs," in *Advances in Inductive Logic Programming*, L. De Raedt, Ed. IOS Press, 1996, pp. 254–264.

[19] T. Sato and Y. Kameya, "Parameter learning of logic programs for symbolic-statistical modeling," *J. Artif. Intell. Res. (JAIR)*, vol. 15, pp. 391–454, 2001.

[20] L. De Raedt, A. Kimmig, and H. Toivonen, "ProbLog: A probabilistic Prolog and its application in link discovery." in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

[21] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: a probabilistic programming language," *Journal of Statistical Software*, 2015.

[22] J. Vennekens, M. Denecker, and M. Bruynooghe, "CP-logic: A language of causal probabilistic events and its relation to logic programming,"

*Theory and Practice of Logic Programming (TPLP)*, vol. 9, no. 3, pp. 245–308, 2009.

[23] J. Vlasselaer, G. Van den Broeck, A. Kimmig, W. Meert, and L. De Raedt, "Anytime inference in probabilistic logic programs with Tp-compilation," in *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015, pp. 1852–1858.

[24] D. Nitti, T. De Laet, and L. De Raedt, "Probabilistic logic programming for hybrid relational domains," *Machine Learning*, vol. 103, pp. 407–449, 2016.