ProbLog2: Probabilistic logic programming

Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens,
Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt

KU Leuven, Belgium,
{firstname.lastname}@cs.kuleuven.be,
https://dtai.cs.kuleuven.be/problog

Abstract. We present ProbLog2, the state of the art implementation of the probabilistic programming language ProbLog. The ProbLog language allows the user to intuitively build programs that do not only encode complex interactions between a large sets of heterogenous components but also the inherent uncertainties that are present in real-life situations. The system provides efficient algorithms for querying such models as well as for learning their parameters from data. It is available as an online tool on the web and for download. The offline version offers both command line access to inference and learning and a Python library for building statistical relational learning applications from the system's components.

Keywords: probabilistic programming, probabilistic inference, parameter learning

1 Introduction

Probabilistic programming is an emerging subfield of artificial intelligence that extends traditional programming languages with primitives to support probabilistic inference and learning. Probabilistic programming is closely related to statistical relational learning (SRL) but focusses on a programming language perspective rather than on a graphical model one. The common goal is to provide powerful tools for modeling of and reasoning about structured, uncertain domains that naturally arise in applications such as natural language processing, bioinformatics, and activity recognition.

This demo presents the ProbLog2 system, the state of the art implementation of the probabilistic logic programming language ProbLog [2–4]. Probabilistic logic programming languages and systems such as ProbLog2, PRISM and CPLint, cf. [1] for an overview, combine ideas from both SRL and probabilistic programming. They are thus related both to SRL systems such as Alchemy and Primula, and to probabilistic programming languages rooted in other programming paradigms, such as Church, Venture, Infer.net, and Figaro.

ProbLog2 supports marginal inference, i.e., computing the conditional probabilities of queries given evidence, parameter learning from data in the form of (partial) interpretations, sampling of possible worlds, and interaction with Python.

ProbLog2 is available on `https://dtai.cs.kuleuven.be/problog`.

2 Language

ProbLog is a probabilistic programming language that extends Prolog along the lines of Sato's distribution semantics. Its development focusses especially on machine learning techniques and implementation aspects.

From a modeling perspective, a ProbLog program has two parts: (1) a probabilistic part that defines a probability distribution over truth values of a subset of the program's atoms, and (2) a logical part that derives truth values of remaining atoms using a reasoning mechanism similar to Prolog. While the latter part simply contains Prolog clauses, the former is specified by *probabilistic facts* $\mathtt{p} :: \mathtt{fact}$, meaning that `fact` is true with probability `p`. All these are probabilistically independent; in case they contain variables, all ground instances are independent as well.

For ease of modeling, ProbLog also allows the use of *annotated disjunctions* $\mathtt{p_1} :: \mathtt{h_1}; \ldots ; \mathtt{p_n} :: \mathtt{h_n} :- \mathtt{body}$ with $\sum_{i=1}^{n} \mathtt{p_i} \leq 1$, meaning that if `body` is true, one of the $\mathtt{h_i}$ will be true according to the specified probabilities $\mathtt{p_i}$. If the probabilities do not sum to one, it is also possible that none of the $\mathtt{h_i}$ is true (with probability $1 - \sum_{i=1}^{n} \mathtt{p_i}$).

The following ProbLog program models a small social network, where people's smoking behaviour is influenced by the behaviour of their friends. And indirectly, also by the behaviour of friends of friends.

```
0.4::asthma(X) :- smokes(X).
0.3::smokes(X).
0.2::smokes(X) :- friend(X,Y), smokes(Y).
friend(1,2). friend(2,1). friend(2,4). friend(3,2). friend(4,2).
```

The main inference task addressed in ProbLog is that of calculating the probability that a query succeeds, for instance `?- asthma(2)` succeeds with a probability 0.15. If we add `asthma(3)` as evidence, the conditional probability of `?- asthma(2)` is 0.19.

3 System blocks

ProbLog2 is the successor of ProbLog1 [2], which was completely integrated in YAP Prolog and performed BDD-based probabilistic inference. ProbLog2 inference consists of a series of transformation steps as shown in Figure 1. The first step is to take the weighted logic program (ProbLog model) and ground it using a Prolog-based grounder. The ground program that is obtained can be represented as a logical formula which may contain cycles. The next steps convert the ground program to a formula in propositional logic, which involves handling cycles. Different options are available for this conversion, using different logical reasoning techniques. Forward compilation directly compiles to sentential decision diagrams (SDD). Alternatively, cycles can be removed first, and the resulting acyclic ground program can then be either transformed into conjunctive normal form and compiled into a d-DNNF, or compiled directly into a SDD.

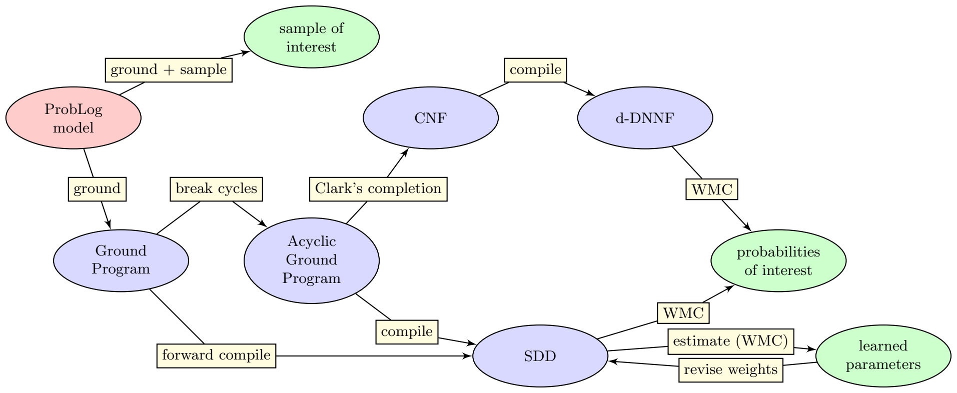


Fig. 1. Overview of the primary ProbLog pipelines.

Both these normal forms support efficient weighted model counting to obtain the final probabilities of interest.

ProbLog also supports query-based sampling in which it assigns a truth value to each of the queries based on their joint probability in the model. This algorithm operates directly during the grounding phase and does not require knowledge compilation.

Parameter learning from interpretations takes a base model and a set of examples as sets of evidence and compiles these into an SDD. These SDDs are then evaluated repeatedly until the weights in the model converge.

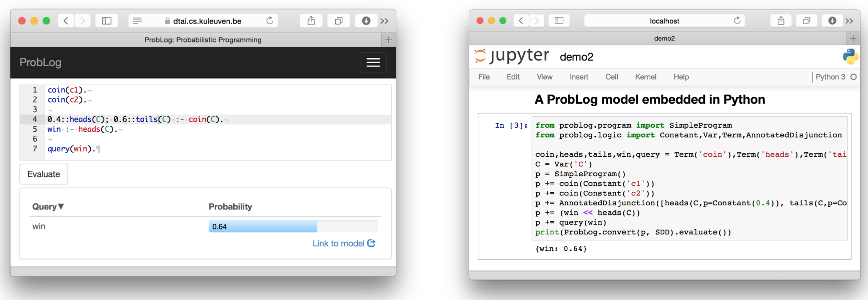
4 System Usage

The ProbLog system can be used through three channels:

Online: The web version of ProbLog allows the user to enter and solve ProbLog problems without the need to install any additional software. It offers an interactive tutorial that illustrates the key modeling concepts in ProbLog through a range of examples, which can be edited on the fly, as well as a separate editor. Examples in the tutorial range from traditional probabilistic models such as Bayesian networks (with plates) and Hidden Markov models to relational probabilistic models that use the full flexibility of ProbLog.

Command line tool: The backend of the online interface is also available as a command line tool, which in contrast to the online version does not impose resource restrictions. It further offers a number of extra options such as, for example, flags to select the required pathway through Fig. 1. It is written in Python and is easy to install on multiple platforms. Additionally, the command line tool allows one to execute ProbLog programs that make use of external functionality written in Python. This makes it possible to, besides the declarative modeling capacities of Prolog, also harness the full power of the Python programming language and its extended ecosystem (e.g. scikit-learn, NLTK). For example, in a natural language processing application, the probability associated with a probabilistic fact `P::similar(d1,d2)` could be defined as the edit-distance between the two string arguments `d1` and `d2` as computed by a corresponding Python function.

Library: ProbLog can be used as a Python package for expressing and querying probabilistic concepts and models. The library provides data structures and algorithms that represent all the components shown in Figure 1. It allows the user to build Statistical Relational Learning (SRL) applications that reuse ProbLog's components.



```
coin(c1).
coin(c2).

0.4::heads(C); 0.6::tails(C) :- coin(C).
win :- heads(C).

query(win).
```

Query▼	Probability
win	0.64

A ProbLog model embedded in Python

```
In [3]: from problog.program import SimpleProgram
        from problog.logic import Constant,Var,Term,AnnotatedDisjunction

        coin,heads,tails,win,query = Term('coin'),Term('heads'),Term('tai
        C = Var('C')
        p = SimpleProgram()
        p += coin(Constant('c1'))
        p += coin(Constant('c2'))
        p += AnnotatedDisjunction([heads(C,p=Constant(0.4)), tails(C,p=Co
        p += (win << heads(C))
        p += query(win)
        print(ProbLog.convert(p, SDD).evaluate())

        {win: 0.64}
```

Fig. 2. The online interface (left) and as a Python library in a Jupyter Notebook (right)

References

1. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Machine Learning (2015)
2. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI) (2007)
3. Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory and Practice of Logic Programming 15(03), 358–401 (2015)
4. Vlasselaer, J., Van den Broeck, G., Kimmig, A., Meert, W., De Raedt, L.: Anytime inference in probabilistic logic programs with Tp-compilation. In: Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI) (2015)