A General View on Probabilistic Logic Programming

Joost Vennekens Sofie Verbaeten*

Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium

Abstract

Probabilistic logic programming offers a way of dealing with both relational and uncertain knowledge. As research in this field has lead to numerous formalisms, originating from diverse origins, it is not always easy to get a clear picture of this domain. In this work, we present a general "framework" for looking at such formalisms. We show how several actual formalisms fit into this view and how this leads to a clearer view on the relationships between them.

1 Introduction and Motivation

One way of dealing with both relational and uncertain knowledge, is by a combination of first order logic and probability theory. Research concerning this topic has not only lead to theoretical probabilistic logics, but also — analogous to logic programming and first order logic — to several more practical "probabilistic logic programming" formalisms. These have originated from a multitude of domains such as logic programming, machine learning and relational databases. However, relatively few contributions have attempted to clarify the relationships between these formalisms beyond the usual unmotivated "formalism X is more general than formalism Y"-claims.

In this work, we offer a first step towards remedying this situation. In section 2, we give an exact definition of our domain of discourse and offer a general overview. In section 3, we present a general description of a probabilistic logic programming "framework". In section 4, we then discuss three actual formalisms and show how they fit into the general framework and what information this can give us about the relationships between them. While research of this kind might not be highly innovative, we believe that, given the current state of the domain of probabilistic logic programming, it does constitute a useful contribution, perhaps even more so than the introduction of "yet another formalism".

*Sofie Verbaeten is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium)(F.W.O. - Vlaanderen).

2 Probabilistic Logic Programming

Before actually going into the details of the field of probabilistic logic programming, it is necessary to be precise about what, in this work, will be considered the definition of this domain. Here, we choose to define a probabilistic logic programming formalism (as opposed to a "probabilistic logic") as one in which each program specifies a *single* probability distribution over some logical entity. For instance, a typical type 2 approach (see below) will express a single probability distribution over a set of possible worlds, while a typical type 1 (see below) approach might express a single probability distribution over the Herbrand universe.

This definition can be motivated by considering the analogy with logic and logic programming. In general, the semantics of a logic (in the mathematical sense) is defined by a relation (usually denoted as $\models$) specifying which mathematical structures are considered models of a certain theory in this logic. Logic programming, however, could be seen as attempting to select a single "intended" model from all mathematical models (e.g. by considering only Herbrand models, making the closed world assumption, etc.).

Nevertheless, this definition is still rather arbitrary. For instance, it excludes, rather ironically, the first ever formalism to explicitly use this term: "Probabilistic Logic Programs" by Subrahmanian and Ng [6]. Indeed, this approach does not attempt to *define* a single probability distribution, but instead uses probability intervals to impose *constraints* on a set of possible distributions. Given the limited size of this contribution, it suits us to exclude such formalisms from consideration.

In the remainder of this section, we present a way to categorize probabilistic logic programming formalisms (as defined above). In [3], Halpern makes some important fundamental observations concerning first order logics of probability. In particular, he distinguishes two different types of semantics for such a logic: the so-called type 1 and type 2 semantics. Traditionally, the intuition behind this distinction is made clear by presenting the two following sentences.

- Type 1: "The probability that a randomly chosen bird flies, is greater than 0.9." (Randomness follows from the process of choosing a bird.)
- Type 2: "The probability that Tweety (a particular bird) flies, is greater than 0.9."(Randomness follows from our lack of knowledge.)

We can therefore make a distinction between formalisms representing the more "objective" type 1 knowledge and those representing the more "subjective" type 2 knowledge. In section 4.3, Stochastic Logic Programs [1, 5], a formalism which expresses type 1 knowledge, will be discussed.

Most of the work concerning probabilistic logic programming focuses on type 2 knowledge. To further categorize these remaining formalisms, we can look at whether they grew out of an attempt to extend logic programming with probability or one to construct a first-order version of a well-known propositional probabilistic framework. These last formalisms allow the representation of an entire "class" of propositional models, from which, for a specific query, an appropriate model can then be constructed "at run-time". Some examples of this approach, usually referred to as Knowledge Based Model Construction, are: Bayesian Logic Programs

of Kersting and De Raedt [4] which will be discussed in section 4.2 and Probabilistic Relational Models by Getoor et al [2].

Finally, the type 2 extensions of logic programming contain formalisms such as Poole's Independent Choice Logic [7], which we will discuss in section 4.1, and Programming In Statistical Modeling by Sato et al [8].

3 General framework

Having defined our domain of discourse, we now give a general description of such a formalism, into which we will, in section 4, fit several actual approaches. Since, as any relevant textbook will show, the concept of the *experiment*[1] lies at the heart of probability theory, it will also feature prominently in our discussion. Indeed, taking such an experiment-centric view, we can formulate the aim of probabilistic logic programming (as defined in section 2) as that of allowing the definition of a single complex experiment (the semantics of the entire program) (e.g. that of determining which of a set of possible world is the "real" one) in terms of a number of smaller experiments (the semantics of some "part" of the program).

More specifically, the semantics of some probabilistic logic program defines a set of experiments. In general, this set could be defined inductively, i.e. the set of possible outcomes or their probability is allowed to depend on the results of some "previous" experiments. Each result of such an experiment has, by definition, a probability. The semantics of the formalism then offers a "combination function"[2], which specifies the probability of each "total result" (i.e. a result for each experiment), in terms of the probabilities of the results of individual experiments. This yields a combined probability space, which then, finally, is mapped into some sort of result space, which fully defines the semantics of the formalism.

As such, the semantics of a probabilistic logic programming formalism can be completely described by stating which experiments it defines, how the probabilities of these simple experiments are combined and how the resulting probability space is translated to a result space.

4 A Detailed Comparison

In this section, we discuss a single formalism chosen from each of the categories presented in section 2. We have based this choice both on our own opinion of which formalism best represents an entire approach and on the availability of literature comparing the chosen formalisms to others; [4] compares BLPs to a number of other formalisms, while [1] does the same for SLPs. We will give a description of each of these formalisms, show how they fit in the terminology of section 3 and attempt to clarify the relationships between them. To illustrate this, we will also represent the Hidden Markov Model (HMM) of figure 1 in each of these formalisms.

[1] By "experiment", we simply mean a probability distribution over some set of possible results.

[2] To avoid confusion, it should be noted that our use of this term is unrelated to combination functions in the context of Bayesian networks.

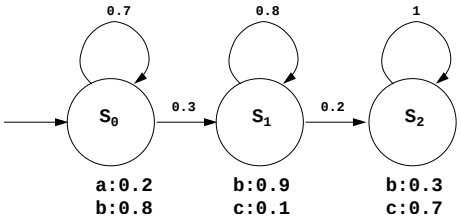

Figure 1: A Hidden Markov Model

4.1 Independent Choice Logic

The Independent Choice Logic (ICL) [7] originated as a probabilistic generalization of abductive logic programming. A program in this logic consists of two parts: A normal logic program F and a set C of declarations of following form $random([a_1 : \alpha_1, \ldots, a_n : \alpha_n])$, with a_i atoms (sometimes called hypotheses or abducibles) and $\alpha_i \in [0,1]$, such that $\sum \alpha_i = 1$. Furthermore, such a pair (C,F) must satisfy following conditions: F must be acyclic, no atom a_i appearing in a *random*-statement in C may unify with another atom a_j which also appears in a *random*-statement in C and no head of a clause in F may unify with an atom a_i appearing in C.

The semantics of an ICL program is defined using the concept of a "total choice", which is simply a selection of an atom a_{ij} from each ground instantiation of a *random*-declaration j. Each such choice corresponds to a possible world, namely the unique stable model of F augmented with the chosen facts a_{ij}. Since total choices are mutually exclusive, there is only one choice for each possible world. The probability of a world is $\prod_j \alpha_{ij}$, where α_{ij} is the probability associated with atom a_{ij} chosen from *random*-declaration j.

So, rephrasing this in our terminology, each *random*-statement defines a single experiment, namely that which chooses an atom from its argument-list. The probabilities α_{ij} are combined into a probability for each total result by assuming independence between the simple experiments, i.e. the "combination function" used is simple multiplication. This combined probability space is than transformed into the "result space", by mapping each set of choices to the unique stable model of the clauses F and the chosen facts.

This discussion shows that, in modeling the HMM of figure 1, the state transitions and possible output will need to be represented by *random*-statements, while the structure of the HMM will be modeled in the logical part:

$$state(S,T) \leftarrow state(Sprev, T-1),\ goto(Sprev, S, T-1).$$
$$out(C,T) \leftarrow state(S,T),\ out(S,C,T).$$
$$state(s0,0). \qquad goto(s2,s2,T).$$
$$random([goto(s0,s1,T) : 0.3, goto(s0,s0,T) : 0.7]).$$
$$random([goto(s1,s2,T) : 0.2, goto(s1,s1,T) : 0.8]).$$
$$random([out(s0,a,T) : 0.2, out(s0,b,T) : 0.8]).$$
$$random([out(s0,b,T) : 0.9, out(s0,c,T) : 0.1]).$$

$random([out(s0, b, T) : 0.3, out(s0, c, T) : 0.7]).$

The interesting part of this formalism, is that the probabilistic and logical part serve completely separate purposes: The *random*-statements define a very simple chance-setup and the logical part serves only to derive the final possible worlds.

4.2 Bayesian Logic Programs

Bayesian Logic Programs [4], or BLPs for short, are based on the (propositional) concept of Bayesian networks. Each ground atom a with predicate-symbol p represents a random variable, which can take on a value from a domain d_p associated with this predicate p. A clause in a BLP is of the form $H \mid A_1, \ldots, A_n$, with H and the A_i atoms. Such a clause requires a conditional probability table (CPT) to be associated with it, specifying the conditional probability of each possible value for the random variables corresponding to the atom in the head, given the values of those corresponding to the atoms in the body. For each predicate p, there is at most one clause with p in its head[3].

The semantics of a BLP is defined by relating it back to a Bayesian network. As previously mentioned, each element of the Herbrand base corresponds to a node. There is an edge from a node representing ground atom X to a node representing ground atom Y if the program contains a clause $c = X' : - \ldots, Y', \ldots$, for which a substitution θ exists, such that $X'\theta = X$ and $Y'\theta = Y$. The CPT associated with such an edge is simply that which was associated with clause c. In order for this semantics to work, this Bayesian network must obviously be well defined, i.e. the number of parents of each node must be finite and, just as in ICL, the program must be acyclic.

So, each clause in a BLP defines a simple experiment, namely that of choosing one possible values for the predicate of the atom in its head. The probability distribution on these possible outcomes obviously depends on the outcome of "previous" experiments, i.e. those corresponding to the atoms in its body. This explains the necessity of the acyclicity condition. The probabilities of these simple experiments are then combined by making the usual independence assumptions of Bayesian networks. The resulting combined probability space gives the semantics of the BLP.

Modeling the HMM of figure 1 by a BLP, is rather straightforward:

$out(T) \mid state(T).$
$state(T) \mid state(T-1).$
$state(0).$

$out(T)$	$state(T) = s_0$	$\cdot = s_1$	$\cdot = s_2$
a	0.2	0	0
b	0.8	0.9	0.3
c	0	0.1	0.7

[3] Technically speaking, a BLP can have multiple clauses for a certain predicate. However, in this case, a combination rule must be given, which specifies how these clauses can be combined to a single clause.

state(0)	
s_0	1
s_1	0
s_2	0

$state(T)$	$state(T-1) = s_0$	$\cdot = s_1$	$\cdot = s_2$
s_0	0.7	0	0
s_1	0.3	0.8	0
s_2	0	0.2	1

Comparing this discussion to that of ICL in section 4.1, we see that the mayor difference between these two formalism lies in the fact that the logical part of a BLP structures the set of experiments it describes (and therefore BLPs use conditional probabilities), while ICL describes a "flat" set of experiments and uses its logical part only in mapping the combined probability space to a result space.

In transforming a BLP to an ICL program, we need one *random*-statement for each possible experiment in the BLP, i.e. one for each possible assignment of values to the atoms in the body of each clause. Therefore, we need to make the value of an atom, which is basically an implicit argument in a BLP, explicit. We then split a BLP clause into a set of clauses, namely one for each possible instantiation of the extra arguments. A such, each of these new clauses needs to describe exactly one *unconditional* ICL experiment. We can accomplish this, by splitting such a clause further, i.e. creating a new clause for each possible instantiation of the extra argument of the atom in its head and differentiating the bodies accordingly by adding a new artificial atom to them. It then suffices to provide an appropriate *random*-statement for these new atoms. In this way, these artificial atoms allow us to simulate the conditional probabilities of a BLP by a set of independent experiments.

Conversely, a definite ICL programs can also be transformed to a BLP. The translation of a *random*-statement is straightforward: For a *random*-statement with n choices, we create a new atom with domain $1, \ldots, n$ and add clauses with degraded CPTs, stating that if this new atom takes on value i then (with probability 1) the ith element from the *random*-statement holds. The logical part of an ICL program can be modeled by giving each of the atoms a domain $\{true, false\}$ and using degraded CPTs to simulate the logical semantics of the clauses, i.e. the head is true if and only if all atoms in the body are true.

4.3 Stochastic Logic Programs

Stochastic Logic Programs [1], or SLPs for short, are a probabilistic generalization of stochastic context free grammars, which "generate" SLD refutations. A SLP consists of a set of range-restricted definite Horn clauses. Each clause c has a label $\lambda_c \in [0, 1]$, which specifies the probability that c is used in an SLD-refutation when an atom with the same predicate as that in the head of c needs to be resolved. As such, denoting by C_p the set of all clauses with a predicate symbol p in the head, the condition that $\sum_{c \in C_p} \lambda_c = 1$ is imposed[4].

SLPs represent type 1 knowledge. More precisely, the probability of each ground instantiation $p(a_1, \ldots, a_n)$ of a predicate p/n is defined as the sum of the probabilities of each proof for $p(a_1, \ldots, a_n)$, normalized by the sum of the

[4] For simplicity, here we only consider "normalized, pure SLPs".

probabilities of each proof for $p(X_1, \ldots, X_n)$. In this formula, the probability of a proof using clause c a number of $t(c)$ times is $\prod_c \lambda_c^{t(c)}$.

In a SLP, a simple experiments is the selection of one clause from all clauses containing a certain predicate in the head. There is one such experiment for each choice-point encountered in a SLD-derivation of a certain query. Therefore, such an experiment obviously depends on the results of previous experiments. The probabilities of simple experiments are once again combined by simple multiplication. The step of going from the combined probability space on SLD-derivations to a result space of possible instantiations of the variables in the query, is done by summing the probability of all refutations which lead to a particular instantiation and then normalizing these sums.

The following SLP gives, for the HMM of figure 1, the desired distribution on the instantiation of the $output/2$ predicate. Note that here we do not need to use a third argument (the time-point) for $out/2$ and $goto/2$.

$1 : output(C,T) : -state(S,T), out(C,S).$
$0.5 : state(S,T) : -T > 0, state(Sprev,T), goto(Sprev,S).$
$0.5 : state(s0,0).$

$0.2/3 : out(a,s0).$	$0.9/3 : out(b,s0).$	$0.3/3 : out(b,s0).$
$0.8/3 : out(b,s0).$	$0.1/3 : out(c,s0).$	$0.7/3 : out(c,s0).$
$0.7/3 : goto(s0,s0).$	$0.8/3 : goto(s1,s1).$	$1/3 : goto(s2,s2).$
$0.3/3 : goto(s0,s1).$	$0.2/3 : goto(s1,s2).$	

Transforming a SLP to an ICL-program, requires constructing a *random*-statement for each possible SLD-choice-point. Therefore, we need to add an extra argument to the atoms, allowing us to differentiate between different "calls" of the same atom. Then, we need to add a *random*-statement for each predicate, containing an atom for each clause with that predicate in its head. These atoms are added to the bodies of the corresponding clause, such that this clause can only by "used" if it is chosen. Finally, the sum of the probabilities of all worlds which contain a specific ground instantiation of a predicate p/n, divided by the sum of the probabilities of all worlds containing *some* instantiation of p/n, gives the semantics of the SLP.

Due to SLPs type 1 nature and the tight coupling of its semantics to SLD resolution, it is currently not clear whether and how one can transform ICL-programs to SLPs.

5 Conclusions and Future Work

In section 3, we presented a general "framework" for probabilistic logic programming formalisms. In section 4, we showed how a probabilistic generalization of abductive logic programming (ICL) and a first-order generalization of Bayesian networks (BLP) and of stochastic context-free grammars (SLP) fit into this view. This allowed us to see past superficial differences in terminology and presentation and clearly determine the differences and similarities between them. ICL uses a

simple set of experiments, but has a complex mapping from combined probability space to its "result space". A BLP defines a more complex set of experiments, but its semantics is simply the combined probability space. A SLP also has a complex set of experiments and, in addition to this, its semantics maps the combined probability space to a distribution over instantiations of the query, giving it its typical type 1 characteristics. In combining the probabilities of simple experiments, all these formalisms resort to assuming independence by default. We were also able to study equivalences between these formalism. ICL turned out to be the most general, with it being possible to represent both BLPs and SLPs in this formalism. Conversely, definite ICL programs can be transformed to BLPs, but the situation with SLPs is less clear.

While we could not, due to space restrictions, make this framework or our comparison of the formalisms mathematically precise, nor offer a fully complete view of the domain, we do believe that the work presented here already shows that doing so in future work would offer an even more significant contribution.

References

[1] J. Cussens. Integrating probabilistic and logical reasoning. *Electronic transactions on Artificial Intelligence*, 3(B):79–103, 1999.

[2] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 7–34. Springer-Verlag, 2001. to appear.

[3] J.Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1989.

[4] K. Kersting and L. De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.

[5] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, page 29. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[6] R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[7] D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

[8] T. Sato and Y. Kameya. PRISM: A language for symbolic-statistical modeling. *Proceedings of IJCAI 97*, pages 1330–1335, 1997.