On the Relationship between Logical Bayesian Networks and Probabilistic Logic Programming based on the Distribution Semantics

Daan Fierens

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, 3001 Heverlee, Belgium,
Daan.Fierens@cs.kuleuven.be

Abstract. A significant part of current research on ILP deals with probabilistic logical models. Over the last decade many logics or languages for representing such models have been introduced. There is currently a great need for insight into the relationships between all these languages. One class of languages are those that extend probabilistic models with elements of logic, such as in the language of Logical Bayesian Networks (LBNs). Some other languages follow the converse strategy of extending logic programs with a probabilistic semantics, often in a way similar to that of Sato's distribution semantics.
In this paper we study the relationship between the language of LBNs and languages based on the distribution semantics. Concretely, we define a mapping from LBNs to theories in the Independent Choice Logic (ICL). We also show that this mapping provides us with a learning algorithm for ICL.

1 Context: Probabilistic ILP and Statistical Relational Learning

The fields of probabilistic inductive logic programming (probabilistic ILP) and statistical relational learning (SRL) have recently witnessed a large interest in probabilistic logical models and languages for representing such models [3]. Several popular languages deal with probabilistic extensions of logic programs. Syntactically one typically uses logic programs in which facts, clauses, or heads of clauses are annotated with probabilities. Semantically one often relies (explicitly or implicitly) on Sato's *distribution semantics (DS)* [10]. We refer to languages that fit this description as *DS languages*. Examples are PRISM [3, Ch.4], the Independent Choice Logic [9], ProbLog [4] and even Logic Programs with Annotated Disjunctions [11]. Other popular languages deal with extensions of probabilistic graphical models to the relational case. For instance, Markov Logic [7, Ch.12] and Relational Markov Networks [7, Ch.6] are based on undirected models, while many other languages are based on directed models: Relational Bayesian Networks [3, Ch.13], Probabilistic Relational Models [7, Ch.5], Bayesian Logic Programs [7, Ch.10], BLOG [7, Ch.13], *Logical Bayesian Networks (LBNs)* [5, 6] and others. In this paper we focus on the language of LBNs, which is strongly related to other languages based on Bayesian networks, especially BLPs and PRMs [5].

2 Problem Statement, Goal and Contributions

The plethora of languages in SRL and probabilistic ILP is sometimes referred to as 'alphabet soup' (consisting of the acronyms of the many languages). There is currently a

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

great need for insight into the relationships between all these languages [2]. The goal of this paper is to study the relationship between LBNs and DS languages. For concreteness, we focus on one particular DS language, namely the *Independent Choice Logic (ICL)* [9], but our discussion largely applies also to each of the other DS languages.

One tool for obtaining insight into the relationships between the various languages is to define translations or mappings between them [3, Ch.12+13][2]. In this paper we show that each LBN can be mapped to an equivalent ICL theory (this is our first contribution). Based on this mapping, we show how the existing learning algorithms for LBNs can serve as a basis for learning ICL theories (second contribution).

We proceed as follows. First we briefly review ICL and LBNs. Then we explain how to map an LBN to an equivalent ICL theory. Finally, we consider the problem of learning ICL theories from data (by means of the algorithms developed for LBNs).

3 Independent Choice Logic (ICL)

In the definitions below we assume the existence of two disjoint sets of atoms: the set of *base atoms* and the set of *derived atoms*. An *alternative* is a (finite) set of base atoms.

An *ICL theory* is a triple $(R, \mathcal{A}, P_0)$, with R an acyclic logic program, $\mathcal{A}$ a set of alternatives[1], and P_0 a probability distribution over each alternative. More formally P_0 is defined as a function that assigns to each atom α in each alternative a number $P_0(\alpha) \in [0, 1]$ such that for each alternative $A \in \mathcal{A}$: $\sum_{\alpha \in A} P_0(\alpha) = 1$. The logic program R is constrained in the sense that the heads of the rules cannot unify with base atoms (only with derived atoms). The set of alternatives $\mathcal{A}$ is constrained in the sense that no atom in any alternative can unify with any other atom in the same or a different alternative. These constraints on R and $\mathcal{A}$ are needed to allow for an 'independent choice' of a base atom from each (grounded) alternative.

The semantics of an ICL theory is that it defines a *probability distribution over possible worlds*. A *possible world* is an interpretation of all (base and derived) atoms. The distribution over possible worlds is derived from the distribution P_0 and the logic program R as follows. A *total choice* is a set of ground base atoms that can be obtained by selecting from each grounding of each alternative in $\mathcal{A}$ exactly one atom. To each total choice C corresponds one possible world. In this world an atom is true if and only if the atom is entailed by C and R, or formally, if it is in the stable model[2] of $C \cup R$. The probability of this world is the same as that of its total choice C: $\prod_{c \in C} P_0(c)$.

The above definitions are essentially those of Poole [9][3, Ch.8]. For the purpose of this paper, we need one extension to these definitions: the extension of ICL with aggregates. Concretely, we allow aggregate literals in the bodies of the clauses of the logic program R (see the example in Section 5). For this we need an extension of the stable model semantics towards logic programs with aggregates. We use the extension of Pelov et al. [8]. One potential complication with using this in ICL is that all stable models in ICL are required to be unique and two-valued [9, p.29]. The stable models of Pelov et al. satisfy these requirements for logic programs that are 'aggregate stratified' [8]. Fortunately, all programs that we consider in this paper are indeed stratified.

[1] In this paper we assume that there is a finite number of alternatives and that R is functor free.

[2] For negation free programs, the stable model is equal to the least Herbrand model.

4 Logical Bayesian Networks (LBNs)

In LBNs [5, 6] we assume that there are some predicates that determine the domain of discourse and that there is no uncertainty about these predicates. For example, in a university domain we could have predicates $student/1$, $course/1$ and $takes/2$. The semantics of an LBN is only defined with respect to a given interpretation of these predicates. We refer to such an interpretation as an *input interpretation* for that LBN.

In LBNs, special predicates are used to represent random variables (RVs). We refer to such predicates and the corresponding atoms as *probabilistic predicates/atoms*. A ground probabilistic atom represents a specific RV, while a non-ground probabilistic atom represents a 'parameterized' RV. Each probabilistic predicate p/n has an associated 'random variable declaration', or briefly *declaration*, which specifies which RVs built from p/n exist for a certain input interpretation. Each probabilistic predicate also has an associated *range*, which specifies the (finite) set of values that these RVs can take. In our university example, probabilistic predicates could be $grade/2$ with declaration $random(grade(S, C)) \leftarrow student(S), course(C), takes(S, C)$ and range $\{good, ok, bad\}$, and $graduates/1$ with declaration $random(graduates(S)) \leftarrow student(S)$ and range $\{yes, no\}$. When given an input interpretation I (that specifies for instance the predicates $student/1$, $course/1$, and $takes/2$), we can use the declarations to obtain the set of all RVs that are defined for I. We denote this set by $\mathcal{RV}(I)$.

Another concept in LBNs is that of a *first-order logical probability tree* for a probabilistic predicate p/n. This is a decision tree in which each internal node contains a boolean test, and each leaf node contains a probability distribution on the range of p/n. The purpose of such a tree is to specify how RVs built from the predicate p/n depend on the other RVs. An example of a tree for $graduates(S)$ is given in Figure 1. As this tree shows, two types of tests can be used in internal nodes. The first type is a test on the value of a parameterized RV such as $grade(S, C)$=*bad*. Free logical variables in such tests are (implicitly) existentially quantified, hence this test checks whether there is a course C for which the given student S has grade 'bad'. The second type is a test on an aggregate function such as $mode(grade(S, C))$=*good*, which checks whether the most frequent grade of S is 'good'.

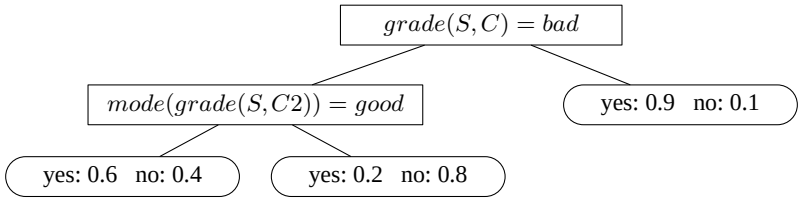

Fig. 1. A probability tree for *graduates*(S). The logical variables C and $C2$ in the tests in internal nodes are called *free*. If a test in a node succeeds, the left branch is taken, otherwise the right.

An *LBN* consists of one declaration and logical probability tree for each probabilistic predicate. Given an input interpretation I, the trees in an LBN determine a dependency relation between the RVs in $\mathcal{RV}(I)$, called the *parent relation*. If this parent relation is acyclic, then we call I a *legal input interpretation* for that LBN.

The semantics of an LBN is that it *maps each legal input interpretation I to a probability distribution over the possible worlds for I*. Each possible world is a joint state of the RVs in $\mathcal{RV}(I)$. The probability of a world is defined as a product of conditional probabilities, like in a Bayesian network. The conditional probability distribution for a particular RV given its parents is defined by the corresponding probability tree.

5 Mapping an LBN to an ICL Theory

Poole already showed that any discrete Bayesian network can be mapped to an ICL theory that specifies the same probability distribution [3, Ch.8]. In this work we essentially extend this propositional result to the first-order case.

Given an LBN, we want to find an 'equivalent' ICL theory. A technical complication is that an ICL theory directly determines a probability distribution, while an LBN maps input interpretations to probability distributions. Hence we define the mapping problem as follows: given an LBN $\mathcal{L}$, find a logic program R, a set of alternatives $\mathcal{A}$, and a probability distribution P_0 over each alternative, such that for any input interpretation I that is legal for $\mathcal{L}$, the probability distribution P_{LBN} of $\mathcal{L}$ for I is equal to the probability distribution P_{ICL} of the ICL theory $(R \cup I, \mathcal{A}, P_0)$. We refer to the triple $(R, \mathcal{A}, P_0)$ as the *equivalent ICL theory* of the LBN.[3] It turns out that finding this is always possible. In other words, **any LBN can be mapped to an equivalent ICL theory**.

To map an LBN to an equivalent ICL theory, we need to map the probability tree (and declaration) of each probabilistic predicate in the LBN. This mapping can be done for each probabilistic predicate separately. In other words, the mapping is local.

To map the probability tree of a probabilistic predicate p/n, we map each path (from the root to a leaf) in the tree to an ICL clause[4] $h \leftarrow b$ and a corresponding alternative A. The head h is a non-ground atom built from the predicate p/m, with m=n+1 (the last argument indicates the value of the corresponding parameterized RV [3, Ch.8]). The body b contains literals describing the path to the leaf and also contains a unique base atom. The alternative A specifies the probability distribution for this base atom, which is the same as the distribution in the considered leaf. While the main lines of this mapping are the same as for mapping a *propositional* probability tree, some complications arise because we are dealing with *first-order logical* trees here. We now illustrate this with an example (space restrictions prevent us from completely formalizing our mapping here).

1. Let us start with the leftmost leaf of the tree for $graduates(S)$ shown in Figure 1. Note that we end up in this leaf if the tests in both internal nodes succeed. This leaf is mapped to the following clause and alternative in the ICL theory.[5]

   $$graduates(S, Val) \leftarrow grade(S, C, bad), mode(G, grade(S, C2, G), good), student(S), b_1(S, Val).$$

   $$\{\ P_0(b_1(S, yes)) = 0.6\ ,\ P_0(b_1(S, no)) = 0.4\ \}$$

   The atom $b_1(S, Val)$ is a base atom. It is important to *not* include C as an argument in this atom: if we would include it (by writing $b_1(S, C, Val)$), then the effect would be that for some S both $graduates(S, yes)$ and $graduates(S, no)$

[3] This is a slight abuse of terminology since the actual ICL theories involve not R but $R \cup I$.

[4] Sometimes additional clauses are needed (to define auxiliary predicates, see below).

[5] For aggregates, such as $mode$, we use syntax similar to that of the *findall*/3 predicate in Prolog.

could become true at the same time (in the same possible world), and this is unwanted because it is not possible in the original LBN. As another issue, note that the body of the clause contains the atom $student(S)$. This comes from the declaration $random(graduates(S)) \leftarrow student(S)$ in the LBN. While in this particular case the condition $student(S)$ is redundant, in general such conditions need to be included to ensure that atoms in the ICL theory only become true when appropriate.

2. The middle leaf in the tree can be mapped in a similar way (note the negated atom).
   $graduates(S, Val) \leftarrow grade(S, C, bad), \neg mode(G, grade(S, C2, G), good),$
   $student(S), b_2(S, Val).$
   $\{\ P_0(b_2(S, yes)) = 0.2\ ,\ P_0(b_2(S, no)) = 0.8\ \}$
3. The rightmost leaf in the tree brings up another issue. We need to express that $grade(S, C, bad)$ has failed, but we cannot simply write $\neg grade(S, C, bad)$ since this would cause 'floundering negation' (since C is a free variable). The standard solution is to introduce an auxiliary predicate that 'hides' C and to negate this.
   $hasBadGrade(S) \leftarrow grade(S, C, bad).$
   $graduates(S, Val) \leftarrow \neg hasBadGrade(S), student(S), b_3(S, Val).$
   $\{\ P_0(b_3(S, yes)) = 0.9\ ,\ P_0(b_3(S, no)) = 0.1\ \}$
   Note that the addition of the condition $student(S)$ to the body is really necessary here: if we do not include it, then $graduates(S, Val)$ might become true for some S that is not a student (but for instance a course).

We would like to stress that the above mapping does not simply map an LBN to whatever ICL theory that assigns to each possible world the same probability as the LBN. The mapped ICL theory in addition also expresses the same conditional independencies, and even the same *context-specific independencies* [6][3, p.230], as the LBN.

6 Learning ICL Theories from Data

We now turn to the problem of learning ICL theories from data. We consider the probabilistic learning from interpretations setting [3, Ch.1]. With learning an ICL theory we mean learning not only its 'parameters' (the distribution P_0) but also its 'structure' (the set of alternatives $\mathcal{A}$ and the clauses in the logic program R). To the best of our knowledge, we are the first to consider the problem of structure learning for ICL.[6]

Poole [3, p.239] recently argued that ICL is a good language for learning because *"being based on logic programming , it can build on the successes of ILP"*, and *"one of the most successful methods for learning Bayesian networks is to learn a decision tree for each variable [...] these decision trees correspond to a particular form of ICL rules"*. In previous work [6] we independently developed learning algorithms for LBNs that follow exactly this approach, except that we effectively integrated the two suggestions of Poole (ILP + trees) since we deal with first-order logical trees in LBNs. Concretely, each of the learning algorithms for LBNs essentially consists of a search algorithm akin to Bayesian networks (such as ordering-search or structure-search) that is wrapped around the ILP system TILDE that learns first-order logical decision trees.

The mapping that we described in the previous section directly leads to a simple way of **learning ICL theories**: we first learn an LBN using one of the existing algorithms,

[6] Parameter learning for ICL has been tackled recently by Carbonetto et al. [1].

and then map the LBN to its equivalent ICL theory. As a proof of concept, and to show what kind of ICL theories can be learned, we applied this approach to the UW-CSE dataset, a well-known SRL benchmark [7, Ch.12]. A description of the learning setting, the learned LBN, and the resulting ICL theory can be found in an online appendix (`http://www.cs.kuleuven.be/~dtai/lbn/ilp09`).

An interesting direction for future research is to implement learning algorithms for ICL directly (rather than by mapping learned LBNs). This is challenging because ICL is more flexible as a language than LBNs. Nevertheless, the current learning algorithms for LBNs provide a good starting point for such research.

7 Conclusion

We showed that there is a strong connection between the language of LBNs and ICL: each LBN can be mapped to an equivalent ICL theory (after inclusion of aggregates in ICL). Based on this connection we argued that the existing learning algorithms for LBNs can serve as a basis for learning ICL theories. While we focussed on ICL, our discussion applies to a large extent also to other languages related to the distribution semantics like ProbLog, PRISM, and even Logic Programs with Annotated Disjunctions.

Acknowledgements. Daan Fierens is supported by the Research Fund K.U.Leuven. This research is also supported by GOA/08/008 'Probabilistic Logic Learning'.

References

1. P. Carbonetto et al. Learning a contingently acyclic, probabilistic relational model of a social network. Technical Report TR-2009-08, University of British Columbia, Department of Computer Science. 2009.
2. L. De Raedt et al. Towards digesting the alphabet-soup of statistical relational learning. NIPS Workshop on Probabilistic Programming, 2008.
3. L. De Raedt et al. *Probabilistic Inductive Logic Programming*, LNCS vol. 4911. Springer, 2008.
4. L. De Raedt et al. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proc. of 20th Int. Joint Conf. on Artificial Intelligence*, pages 2462–2467, 2007.
5. D. Fierens. Logical Bayesian networks. Chapter 3 of *Learning Directed Probabilistic Logical Models from Relational Data.* PhD Thesis, Katholieke Universiteit Leuven, 2008. See `http://hdl.handle.net/1979/1833`.
6. D. Fierens et al. Learning directed probabilistic logical models: Ordering-search versus structure-search. *Annals of Mathematics and Artificial Intelligence*, 2009. In press (available online: `http://dx.doi.org/10.1007/s10472-009-9134-9`).
7. L. Getoor and B. Taskar. *An Introduction to Statistical Relational Learning*. MIT Press, 2007.
8. N. Pelov et al. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming*, 7(3):301–353, 2007.
9. D. Poole. Abducing through negation as failure: Stable models within the independent choice logic. *Journal of Logic Programming*, 44(1–3):5–35, 2000.
10. T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proc. of 12th Int. Conf. on Logic Programming*, pages 715–729. MIT Press, 1995.
11. J. Vennekens et al. Logic programs with annotated disjunctions. In *Proc. of 20th Int. Conf. on Logic Programming*, LNCS vol. 3132, pages 431–445. Springer, 2004.