

Mapping Logical Bayesian Networks to Probabilistic Logic Programs with Distribution Semantics

Daan Fierens

Report CW 563, August 2009



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Mapping Logical Bayesian Networks to Probabilistic Logic Programs with Distribution Semantics

Daan Fierens

Report CW563, August 2009

Department of Computer Science, K.U.Leuven

Abstract

A significant part of current research on (inductive) logic programming 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 kind of languages are those that extend probabilistic models with elements of logic, such as 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 define a mapping from LBNs to probabilistic logic programs with the distribution semantics.

Keywords : probabilistic logic programming, Bayesian networks.

CR Subject Classification : I.2.4, G.3

1 Introduction

In the fields of *probabilistic inductive logic programming (PILP)* [3] and *statistical relational learning (SRL)* [8] there is a large interest in probabilistic logical models. Over the last decade many different languages for representing such models have been introduced.

One class of languages deals 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)* [15]. We refer to languages that fit this description as *DS languages*. Examples of DS languages are PRISM [3, Ch.5], the Independent Choice Logic [13], ProbLog [4] and Logic Programs with Annotated Disjunctions [16].

Another popular class of languages deals with extensions of probabilistic graphical models to the relational case. For instance, Markov Logic [8, Ch.12] and Relational Markov Networks [8, 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 [8, Ch.5], Bayesian Logic Programs [8, Ch.10], BLOG [8, Ch.13], CLP(\mathcal{BN}) [3, Ch.6], *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].

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

One tool for obtaining insight into the relationships between 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 Independent Choice Logic (ICL) theory. We also show that there also exist mappings from LBNs to the other DS languages (ProbLog, PRISM and Logic Programs with Annotated Disjunctions) and that these mappings are very similar to the one for ICL. In recent related work we have used these mappings as the basis for a learning algorithm for DS languages [7].

We proceed as follows. In Section 2 we briefly review ICL and LBNs. In Section 3 we discuss the mapping from LBNs to ICL. In Section 4 we discuss the mapping from LBNs to other DS languages. In Section 5 we conclude.

2 Background

We assume familiarity with the basics of logic programming [10] and Bayesian networks [11].

2.1 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 *ICL theory* is a pair (R, \mathcal{A}) with R an acyclic logic program and \mathcal{A} a set of annotated alternatives. An *annotated alternative* is a finite set of base atoms each annotated with a probability (to be precise, each atom α is annotated with a number $P_0(\alpha) \in [0, 1]$ such that $\sum_{\alpha} P_0(\alpha) = 1$). The logic program R in an ICL theory is constrained in the sense that the heads of the clauses cannot unify with base atoms (only with derived atoms). The set of annotated 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) annotated alternative. In this paper we assume that there is a finite number of annotated alternatives in \mathcal{A} and that R is functor free.

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 annotated alternatives \mathcal{A} 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 annotated alternative in \mathcal{A} exactly one atom α . The probability of a total choice C is defined as $\prod_{\alpha \in C} P_0(\alpha)$ with $P_0(\alpha)$ the probability of α according to the annotated alternative that it was selected from. To each total choice C corresponds one possible world: the world in which an atom is true if and only if it is entailed by C and R , i.e., if it is in the stable model of $C \cup R$.¹ The probability of this world is the same as that of its total choice C .

The above definitions are essentially those of Poole [13][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 3.2). For this we need an extension of the stable model semantics towards logic programs with aggregates. We use the extension of Pelov et al. [12]. One potential complication with using this in ICL is that all stable models in ICL are required to be unique and two-valued [13, p.29]. The stable models of Pelov et al. satisfy these requirements for logic programs that are ‘aggregate stratified’ [12]. Fortunately, all programs that we consider in this paper are indeed stratified.

2.2 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* with the obvious meaning. 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 has an associated ‘*random variable declaration*’, which specifies which RVs built from p 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 random variable 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 . 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 . The purpose of such a tree is to specify how RVs built from the predicate p depend on the other RVs. An example of a tree for *graduates/1* is given in Figure 1 (the parameterized RV $graduates(S)$ is called the ‘target’ of this tree). 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 $grade(S, C)=bad$. Logical variables that occur in such tests but not in the target of the tree, such as C , are (implicitly) existentially quantified. Hence the test $grade(S, C)=bad$ checks whether there exists a course C for which the given student S has grade ‘bad’.
- The second type is an aggregate test such as $mode(grade(S, C2))=good$. Logical variables that occur in such tests but not in the target of the tree, such as $C2$, are aggregated

¹For negation free programs the stable model is equal to the least Herbrand model.

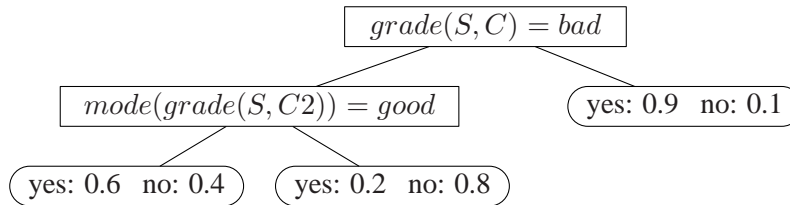


Figure 1: An example of a first-order logical probability tree for the target $graduates(S)$. If a test in a node succeeds, the left branch is taken, otherwise the right branch.

over. Hence the test $mode(grade(S, C2))=good$ checks whether the most frequent grade of S over all courses $C2$ is ‘good’.

An *LBN* consists of one random variable declaration and one 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 $\mathcal{RV}(I)$ (the so-called ‘parent relation’ [5]). If this dependency 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.

3 The Mapping from LBNs to ICL

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 section we essentially extend this propositional result to the first-order case.

3.1 Problem Definition

Given an LBN, we want to find an ‘equivalent’ ICL theory. A technical complication is that an ICL theory directly defines a probability distribution, while an LBN maps input interpretations to probability distributions.² Hence we define the mapping problem as follows: given an LBN L , find a logic program R and a set of annotated alternatives \mathcal{A} such that for any input interpretation I that is legal for L , the probability distribution P_{LBN} of L for I is equal to the probability distribution P_{ICL} of the ICL theory $(R \cup I, \mathcal{A})$. We refer to the pair (R, \mathcal{A}) as the *equivalent ICL theory* of the LBN (this is a slight abuse of terminology since the actual ICL theories involve not R but $R \cup I$).

3.2 The Mapping

In this section we show that (and how) **each 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 random variable 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. Mapping the probability tree for a probabilistic predicate p is essentially done by mapping each path from the root to a leaf in the tree to a corresponding ICL clause and annotated alternative. In addition, some auxiliary clauses (not specific to any leaf in the tree) might need to be added to the ICL theory as well.

²This is a recurring problem when defining mappings between PILP/SRL languages [3, Ch.13].

Concretely, the procedure for mapping an LBN to an equivalent ICL theory is as follows. We start from the empty ICL theory. We loop over all probabilistic predicates in the LBN. For each considered predicate p we perform two steps.

1. As a preprocessing step we annotate the probability tree for p with some information needed in the next step. In the same process we also create some necessary auxiliary clauses and add them to the ICL theory.
2. We loop over all leaves of the annotated tree for p . For each considered leaf we create a corresponding clause and annotated alternative, and add them to the ICL theory.

We now explain these two steps in detail.

3.2.1 Step1: Annotating the Tree and Creating Auxiliary Clauses

3.2.1.1 Main Ideas

The first step is a preprocessing step in which we annotate the considered logical probability tree with some information needed in the next step. Concretely, the goal of the first step is to find for each node in the tree (leaves and internal nodes) a conjunction of literals that describes the path from the root to that node. Below we show that this can be done with a recursive procedure.

Annotating the tree sometimes requires the introduction of new predicates (that are not used in the original probability tree). This is the case whenever the tree contains a non-aggregate test that contains a logical variable that does not occur in the target of the tree, such as the test $grade(S, C) = bad$ in Figure 1 (the variable C does not occur in the target). We call such tests ‘*existential tests*’ because the variables that do not occur in the target (like the variable C) are existentially quantified (recall Section 2.2). We call the new predicates that need to be introduced when annotating a tree with existential tests *auxiliary predicates*. Of course, the definitions of these auxiliary predicates need to be added to the ICL theory that is obtained after mapping the tree. The reason why dealing with existential tests in a tree requires auxiliary predicates is related to negation, see Section 3.2.2.2 (or see Blockeel [1, Section 5.3.2] for more details).

The procedure that we execute in Step1 for the given probability tree T is shown in Figure 2. This procedure annotates the nodes of T , introduces auxiliary predicates, and creates the auxiliary clauses that define these predicates. This procedure is inspired on the DERIVE_LOGIC_PROGRAM procedure of Blockeel [1, Section 5.3.2] for mapping a logical decision tree to a logic program.³ We explain our procedure further by means of an example.

3.2.1.2 Worked Example

Consider the tree of Figure 1. Figure 3 shows the annotated version of this tree. We now explain how this is obtained by following our PREPROCESS procedure of Figure 2.

- We start by annotating the root of the tree. The current value of the parameter $Conj$ in the PREPROCESS procedure is *true*. Hence, we annotate the root with *true*. This indicates that the root node is always reached.
- Next we have to convert the test in the root ($grade(S, C) = bad$). The idea is simply to convert tests that are formulated in terms of parameterized RVs to a more standard form, namely a conjunction of first order literals. This is done by giving each probabilistic predicate in the LBN an extra argument denoting the value of the corresponding parameterized RV. Hence, the test $grade(S, C) = bad$ is converted to the literal $grade(S, C, bad)$.

³In comparison to Blockeel’s procedure our procedure creates fewer auxiliary predicates: our procedure only creates them when strictly necessary, namely for internal nodes with an existential test, while Blockeel creates them for every internal node.

```

procedure PREPROCESS( $T$ ) {
//  $T$  is a first-order logical probability tree
  PREPROCESS( $T, true$ )
}

procedure PREPROCESS( $T, Conj$ ) {
//  $T$  is a first-order logical probability tree
//  $Conj$  is a conjunction of literals
  if  $T$  is a leaf then
    annotate  $T$  with  $Conj$ 
  else
    let  $Root$  denote the root node of  $T$ 
    let  $Test_{root}$  denote the test in  $Root$ 
    let  $L$  (resp.  $R$ ) denote the left (resp. right) subtree of  $T$ 
    annotate  $Root$  with  $Conj$ 
    convert  $Test_{root}$  to a conjunction  $Conj_{root}$  of literals
    if  $Test_{root}$  is an existential test then
      introduce a new unique auxiliary predicate and corresponding atom  $AuxAtom$ 
      create a clause  $AuxAtom \leftarrow Conj \wedge Conj_{root}$ .
      PREPROCESS( $L, Conj \wedge Conj_{root}$ ) // left subtree = succeeding branch
      PREPROCESS( $R, Conj \wedge \neg AuxAtom$ ) // right subtree = failing branch
    else
      PREPROCESS( $L, Conj \wedge Conj_{root}$ )
      PREPROCESS( $R, Conj \wedge \neg Conj_{root}$ )
  }
}

```

Figure 2: Procedure for preprocessing a logical probability tree (this procedure annotates the nodes of the tree and creates clauses that define the auxiliary predicates).

- Next we check whether the test in the root is an existential test. This is indeed the case: the variable C is existentially quantified. Hence, we need to introduce a new auxiliary predicate. The arguments of this auxiliary predicate are the arguments in the considered test that are not existentially quantified; in this case this is only S , hence we introduce a unary auxiliary predicate $aux/1$. The current value of the parameter $Conj$ is $true$ and the value of $Conj_{root}$ is $grade(S, C, bad)$. Hence we have to create a clause $aux(S) \leftarrow grade(S, C, bad)$.
- Now we recursively apply the PREPROCESS procedure to the two subtrees.
 - We start with the left subtree. When the procedure PREPROCESS is called for this subtree, the input parameter $Conj$ in the procedure has the value $grade(S, C, bad)$. Hence, we annotate the root node of the left subtree with $grade(S, C, bad)$. This indicates that this node is reached if and only if $grade(S, C, bad)$ is true.
 - Next we have to convert the test in this node ($mode(grade(S, C2)) = good$). This yields the literal $mode(G, grade(S, C2, G), good)$. Note that for aggregates we use a syntax like that of the $findall/3$ predicate in Prolog.
 - Next we check whether the considered test ($mode(grade(S, C2)) = good$) is existential. This is not the case (aggregate tests are never existentially quantified since all free variables in such tests are aggregated over). Hence no auxiliary predicate needs to be introduced.

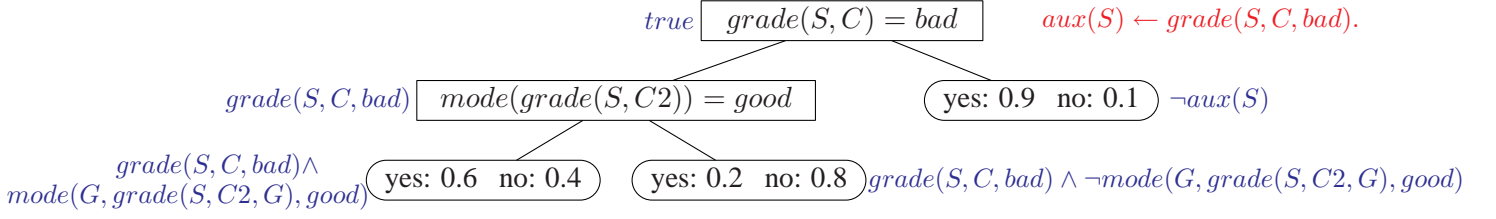


Figure 3: Annotated version of the probability tree shown in Figure 1. The target of this tree is $graduates(S)$. Conjunctions associated with nodes are shown in blue, auxiliary clauses in red.

- Now we recursively apply the PREPROCESS procedure to the two subtrees of the left subtree.
 - * We start with the left subtree of the left subtree; this is the leftmost leaf of the entire tree. The input parameter $Conj$ now has the value $grade(S, C, bad) \wedge mode(G, grade(S, C2, G), good)$. Hence we annotate this leaf with this conjunction. This indicates that this leaf is reached if and only if this conjunction is true.
 - * Then we consider the right subtree of the left subtree. This is the middle leaf of the entire tree. The input parameter $Conj$ now has the value $grade(S, C, bad) \wedge \neg mode(G, grade(S, C2, G), good)$ (note the negated atom). Hence we annotate this leaf with this conjunction.
- Now we are at the right subtree of the entire tree; this is the rightmost leaf. The input parameter $Conj$ has the value $\neg aux(S)$. Hence we annotate this leaf with $\neg aux(S)$. Since we have now traversed the entire tree the procedure terminates here.

3.2.2 Step2: Mapping the Leaves

3.2.2.1 Main Ideas

In the second step we map the annotated tree obtained in the previous step to a set of corresponding ICL clauses and annotated alternatives. This is done by looping over all leaves of the annotated tree. For each considered leaf we create a corresponding ICL clause $h \leftarrow b$ and annotated alternative A .

- The head h is the target atom of the considered tree, again extended with one extra argument denoting the value of the corresponding parameterized RV.
- The body b is a conjunction of literals and consists of three parts:
 - the literals associated with the considered leaf l in the annotated tree (recall that these literals describe the path from the root to l),
 - the literals in the body of the random variable declaration of the target of the considered tree,
 - a unique base atom with the same arguments as the head h .
- The annotated alternative A describes a probability distribution for the above base atom. This distribution is the same as the distribution in the considered leaf of the tree.

We explain this further with an example.

3.2.2.2 Worked Example

Consider the annotated tree constructed in the previous section (the tree in Figure 3). This tree has three leaves.

- Let us start with the *leftmost leaf*. The corresponding ICL clause for this leaf is the following.

$$\begin{aligned} \text{graduates}(S, Val) \leftarrow & \text{grade}(S, C, \text{bad}), \text{mode}(G, \text{grade}(S, C2, G), \text{good}), \\ & \text{student}(S), b_1(S, Val). \end{aligned}$$

- The first two literals in the body of this clause are the literals that are associated with the considered leaf in the annotated tree of Figure 3. These two literals describe the path from the root to this leaf (note that we end up in this leaf if the tests in both internal nodes of the tree succeed).
- The third literal, $\text{student}(S)$, comes from the body of random variable declaration of the target of this tree in the LBN ($\text{random}(\text{graduates}(S)) \leftarrow \text{student}(S)$). While in this particular case the condition $\text{student}(S)$ is redundant, in general such conditions need to be included to ensure that atoms in the ICL theory only become true when appropriate (when they are properly typed, etc., see below).
- The fourth literal in the body, $b_1(S, Val)$, is a unique base atom. It is important that this atom has exactly the same arguments as the head of the clause. In particular, 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 we would get a kind of noisy-or effect with as a result that for some S both $\text{graduates}(S, \text{yes})$ and $\text{graduates}(S, \text{no})$ could become true in the same possible world, and this is unwanted because it is not possible in the original LBN.

The corresponding annotated alternative for this leaf is the following.

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

This specifies a probability distribution for the base atom introduced above. This distribution is the same as the one in the considered leaf of the tree ($\text{yes} : 0.6, \text{no} : 0.4$).

- The mapping of the *middle leaf* in the tree is obtained in a very similar way.

$$\begin{aligned} \text{graduates}(S, Val) \leftarrow & \text{grade}(S, C, \text{bad}), \neg \text{mode}(G, \text{grade}(S, C2, G), \text{good}), \\ & \text{student}(S), b_2(S, Val). \end{aligned}$$

$$\{ P_0(b_2(S, \text{yes})) = 0.2, P_0(b_2(S, \text{no})) = 0.8 \}$$

- The mapping of the *rightmost leaf* is also very similar.

$$\text{graduates}(S, Val) \leftarrow \neg \text{aux}(S), \text{student}(S), b_3(S, Val).$$

$$\{ P_0(b_3(S, \text{yes})) = 0.9, P_0(b_3(S, \text{no})) = 0.1 \}$$

Note that the addition of the condition $\text{student}(S)$ (originating from the random variable declaration $\text{random}(\text{graduates}(S)) \leftarrow \text{student}(S)$ in the LBN) to the body of the clause is really necessary here: if we do not include it, then $\text{graduates}(S, Val)$ can become true for some S that is not a student (but for instance a course).

The mapping of this leaf also illustrates the necessity of introducing the auxiliary predicate $\text{aux}/1$. One might be tempted to not introduce this predicate and write the above clause as $\text{graduates}(S, Val) \leftarrow \neg \text{grade}(S, C, \text{bad}), \text{student}(S), b_3(S, Val)$. However, this would cause a problem with floundering negation since C is a free variable in this clause. This problem does not occur when using the predicate $\text{aux}/1$ (which is defined as $\text{aux}(S) \leftarrow \text{grade}(S, C, \text{bad})$) since this ‘hides’ the variable C .

3.2.3 Summary of the Running Example

In the previous sections we used the probability tree for $graduates(S)$ shown in Figure 1 as the running example. All ICL clauses and annotated alternatives that result from mapping this tree have already been shown in one of the previous sections. For the sake of clarity we now show all these clauses and annotated alternatives again. We rename the auxiliary predicate $aux/1$ into $hasBadGrade/1$ since this name captures the intuitive meaning of this predicate.

$$\begin{aligned}
graduates(S, Val) &\leftarrow grade(S, C, bad), mode(G, grade(S, C2, G), good), \\
&\quad student(S), b_1(S, Val). \\
graduates(S, Val) &\leftarrow grade(S, C, bad), \neg mode(G, grade(S, C2, G), good), \\
&\quad student(S), b_2(S, Val). \\
graduates(S, Val) &\leftarrow \neg hasBadGrade(S), student(S), b_3(S, Val). \\
hasBadGrade(S) &\leftarrow grade(S, C, bad). \\
\{ P_0(b_1(S, yes)) = 0.6, P_0(b_1(S, no)) = 0.4 \} \\
\{ P_0(b_2(S, yes)) = 0.2, P_0(b_2(S, no)) = 0.8 \} \\
\{ P_0(b_3(S, yes)) = 0.9, P_0(b_3(S, no)) = 0.1 \}
\end{aligned}$$

3.2.4 Correctness of the Mapping

In Appendix A we show that the above mapping is correct, i.e., that it yields an ICL theory that is indeed equivalent to the given LBN. As is required by the ICL semantics, this ICL theory is always acyclic, or at least ‘contingently acyclic’ [13, p.30]. This is because, for legal input interpretations, the dependency relation specified by an LBN is acyclic as well.

Given an LBN, let Σ denote the set of predicates of this LBN. If we map this LBN to an ICL theory using the above mapping, then the resulting ICL theory will use a set of predicates $\Sigma \cup \Delta$, where Δ contains all the predicates that are introduced as a by-product of the mapping, namely auxiliary predicates (Section 3.2.1.1) and base predicates (Section 3.2.2.1). For instance, in our running example Δ is the set $\{hasBadGrade/1, b_1/2, b_2/2, b_3/2\}$. Hence, the LBN defines a probability distribution over possible worlds determined by Σ , whereas the ICL theory defines a probability distribution over possible worlds (i.e., interpretations) determined by $\Sigma \cup \Delta$. If we are not interested in the predicates in Δ , then we can of course derive from the ICL theory a probability distribution over interpretations for Σ only. This can be done in the standard way, namely by marginalisation: we sum out all predicates in Δ .⁴ When we say that “the probability distribution of the ICL theory obtained from our mapping is equal to the probability distribution of the given LBN” we are referring to the *marginalised* distribution of the equivalent ICL theory.

3.3 Discussion

We want to stress that the above mapping does not simply map an LBN to whatever ICL theory that defines the same probability distribution as the LBN. The clauses in the mapped ICL theory in addition also express the same conditional independencies, and even the same *context-specific independencies* [6][3, p.230], as the probability trees in the LBN.

Note that for LBNs that do not contain aggregate tests, the mapped ICL theories do not contain aggregates either. Hence, in such cases we can use the original definition of ICL by Poole [13]. When considering LBNs with aggregate tests, the mapped ICL theories also contain aggregates and we resort to the extension of ICL with aggregates as discussed in Section 2.1.

We have seen that each LBN can be mapped to an equivalent ICL theory. This means that the mapping is total. On the other hand, given an ICL theory, one can also ask the question whether there exists an LBN that maps to it. This question is relevant in the context of learning. In recent

⁴Let I_Σ be an interpretation of the predicates in Σ . Then we have that $P(I_\Sigma) = \sum_{I_\Delta} P(I_\Sigma \wedge I_\Delta)$, where the sum is over all possible interpretations I_Δ of the auxiliary predicates in Δ .

related work [7] we have learned ICL theories from data by learning LBNs and mapping them to ICL theories. With this approach we can of course only learn ICL theories for which there exists an LBN that maps to them. We call such ICL theories *LBN-compliant*. LBN-compliant ICL theories all have a particular structure (similar to that of the ICL theory for our running example shown in Section 3.2.3). However, there exist valid ICL theories that do not have this structure and hence are not LBN-compliant (and cannot be learned with our approach [7]). This particular structure that we are talking about is visible in both parts of the ICL theories: the clauses and the annotated alternatives.

- For each *annotated alternative* in an LBN-compliant ICL theory it holds that all atoms in the annotated alternative 1) are built from the same predicate, and 2) have the same non-ground arguments, with the exception of the last argument which is ground and is different for each atom in that alternative. An example of such an annotated alternative is $\{outcome(Toss, heads) = 0.5, outcome(Toss, tails) = 0.5\}$. In contrast, an annotated alternative like $\{heads(Toss) = 0.5, tails(Toss) = 0.5\}$ does not have this structure, hence any ICL theory containing this alternative is not LBN-compliant.
- The *clauses* in an LBN-compliant ICL theory also have a particular structure. Most important is that the bodies of all clauses with a unifiable head have a particular structure: they correspond to the paths in a logical decision tree, i.e., these bodies are ‘*tree-structured*’. If the bodies of a set of clauses are tree-structured, this implies that:
 - The bodies are *mutually exclusive*, i.e., in any possible world at most one of them is true (this is because all paths in a decision tree are mutually exclusive). Hence, any ICL theory that contains two clauses with a unifiable head and bodies that are not mutually exclusive is not LBN-compliant; this includes for instance ICL theories with *noisy-or* dependencies [3, Ch.8].
 - The bodies are *covering*, i.e., in any possible world at least one of them is true.

Note, however, that it is not the case that any set of bodies that are mutually exclusive and covering is necessarily tree-structured: some sets of bodies are mutually exclusive and covering but yet do not correspond to the paths of any decision tree (even in the propositional case [14, Section 3.2]).

4 The Mapping from LBNs to LPADs, PRISM and ProbLog

In the previous section we have shown that each LBN can be mapped to an equivalent ICL theory. In this section we show that there also exist mappings from LBNs to PRISM, ProbLog and LPADs (Logic Programs with Annotated Disjunctions), and that these mappings are very similar to the one for ICL.

To make the mappings to LPADs, ProbLog and PRISM total (i.e., to ensure that each LBN can be mapped to an equivalent LPAD, PRISM program or ProbLog program) we need to extend these languages with the concept of aggregates in the same way as we did for ICL. As for ICL, it holds that aggregates are needed only when the considered LBN also contains aggregate tests, otherwise the LBN can be mapped to an equivalent LPAD, PRISM program or ProbLog program without aggregates. For LPADs, the possibility of an extension with aggregates has been mentioned before [16]. For PRISM and ProbLog, which are based on Prolog, aggregates could be defined in terms of meta-predicates like *findall/3* but as far as we know this is not yet supported by neither PRISM nor ProbLog.

In the context of mapping LBNs the issue of aggregates is orthogonal to the differences between the various DS languages (ICL, LPADs, PRISM and ProbLog). Hence we do not discuss this issue any further in the following sections and assume that the languages of LPAD, PRISM and ProbLog are extended with aggregates.

4.1 The Mapping from LBNs to Logic Programs with Annotated Disjunctions (LPADs)

For an introduction to LPADs, see Vennekens et al. [16]. Note that the language of *CP-logic* is equivalent to LPADs [17]. Hence the comments below apply equally well to CP-logic as to LPADs.

It is known that ICL is strongly related to LPADs [17, Section 9.4.1]. As a consequence, **each LBN can be mapped to an equivalent LPAD** in almost exactly the same way as discussed for ICL.

- The *clauses* in the equivalent LPAD are exactly the same as those in the equivalent ICL theory.
- The *probability distribution over base atoms*, which is modelled in ICL by means of annotated alternatives, is modelled in LPADs using clauses with a true ('empty') body. There is a direct correspondence between ICL alternatives and such clauses. For instance, the ICL alternative $\{ P_0(b_1(S, yes)) = 0.6, P_0(b_1(S, no)) = 0.4 \}$ corresponds to the LPAD clause $b_1(S, yes) : 0.6 \vee b_1(S, no) : 0.4$.

There exists at least one other way of mapping LBNs to LPADs (yielding different but equivalent LPADs). Because this mapping is complicated by some rather technical issues, we do not discuss it further.

4.2 The Mapping from LBNs to PRISM

For an introduction to PRISM, see Sato and Kameya [3, Ch.5]. The PRISM language is strongly related to ICL. There are some differences in the assumptions made by the two languages but these differences do not have an impact on the mapping of LBNs, see below. Apart from the different assumptions, the main differences between ICL and PRISM are syntactical. Hence, **each LBN can be mapped to an equivalent PRISM program** in a very similar way as discussed for ICL.

- The *clauses* in the equivalent PRISM program are the same as those in the equivalent ICL theory.
- The *probability distribution over base atoms*, which is modelled in ICL by means of annotated alternatives, is modelled in PRISM by means of *multi-ary random switches*. Each ICL alternative can be mapped straightforwardly into an equivalent multi-ary random switch by means of the `msw/3` construct in PRISM [3, Ch.5, Section 2.1].

As mentioned, there are some differences in the assumptions made by PRISM and ICL. On the one hand there are some assumptions made by ICL but not by PRISM (namely about the finiteness of the theories/programs [3, Ch.5, Section 2.1]). These assumptions are irrelevant in the context of mapping LBNs: our main result says that each LBN can be mapped to an equivalent ICL theory; obviously if some restrictions of ICL are lifted, the mapping remains valid. On the other hand, there are some assumptions made by PRISM but not by ICL [3, Ch.5, Section 2.2]. The main such assumption in PRISM is the '*exclusiveness condition*' which requires that the different explanations of an atom are mutually exclusive. Fortunately, any PRISM program obtained by mapping an LBN automatically satisfies this condition: the bodies of all clauses with unifiable heads are mutually exclusive. This holds because these bodies correspond to the different paths in a probability tree, and all paths in a decision tree are indeed mutually exclusive.

4.3 The Mapping from LBNs to ProbLog

For an introduction to ProbLog, see De Raedt et al. [4]. When originally introduced, the ProbLog language did not include the concept of negation. However, in recent work [9] ProbLog has been extended so as to deal with negative literals in clause bodies as well. The main remaining difference between ProbLog and ICL is that ProbLog does not have the concept of annotated alternatives. Instead, ProbLog simply associates probabilities with facts. However, ICL-like alternatives can be simulated in ProbLog (see below). As a consequence, the languages of ICL and ProbLog are practically equivalent.

Given the close link between ICL and ProbLog, it is not surprising that **each LBN can be mapped to an equivalent ProbLog program** in a very similar way as discussed for ICL.

- The *clauses* in the equivalent ProbLog program are exactly the same as those in the equivalent ICL theory.
- The *probability distribution over base atoms* (‘probabilistic atoms’ in ProbLog terminology), which is modelled in ICL by means of annotated alternatives, is obtained in ProbLog by means of probabilistic facts and the use of negation. For instance, the ICL alternative $\{ P_0(b_1(S, yes)) = 0.6, P_0(b_1(S, no)) = 0.4 \}$ is modelled in ProbLog using the following probabilistic fact and clause.

```
0.6 : b1(S, yes).  
b1(S, no) :- not b1(S, yes).
```

This indeed specifies the same probability distribution over the predicate $b_1/2$ as the ICL alternative does.

The above shows that binary alternatives can be simulated in ProbLog. For multi-ary alternatives this is possible as well but slightly more complicated, see the example of Kimmig et al. [9, Section 2].

5 Conclusion

We showed that there is a strong connection between the language of Logical Bayesian Networks and languages based on the distribution semantics. Concretely, we showed that each LBN can be mapped to an equivalent Independent Choice Logic theory (after extension of the ICL language with aggregates). We also showed that there exist mappings from LBNs to ProbLog, PRISM and Logic Programs with Annotated Disjunctions, and that these mappings are very similar to the one for ICL. In related work we have used these mappings as the basis for a learning algorithm for languages based on the distribution semantics [7].

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] H. Blockeel. *Top-down Induction of First Order Logical Decision Trees*. PhD Thesis, Katholieke Universiteit Leuven, 1998. <https://lirias.kuleuven.be/handle/123456789/196276>
- [2] L. De Raedt, B. Demoen, D. Fierens, B. Gutmann, G. Janssens, A. Kimmig, N. Landwehr, T. Mantadelis, W. Meert, R. Rocha, V. Santos Costa, I. Thon, J. Vennekens. Towards digesting the alphabet-soup of statistical relational learning. NIPS Workshop on Probabilistic Programming, 2008.

- [3] L. De Raedt, P. Frasconi, K. Kersting, and S.H. Muggleton. *Probabilistic Inductive Logic Programming*. Lecture Notes in Computer Science, vol. 4911. Springer, 2008.
- [4] 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*, 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. <http://hdl.handle.net/1979/1833>
- [6] D. Fierens, J. Ramon, M. Bruynooghe, and H. Blockeel. Learning directed probabilistic logical models: Ordering-search versus structure-search. *Annals of Mathematics and Artificial Intelligence*, 54(1-3):99–133, 2008.
- [7] D. Fierens. On the Relationship between logical Bayesian networks and probabilistic logic programming based on the distribution semantics. In *Postproceedings of the 19th International Conference on Inductive Logic Programming*. Lecture Notes in Computer Science. Springer, 2009. In press.
- [8] L. Getoor and B. Taskar. *An Introduction to Statistical Relational Learning*. MIT Press, 2007.
- [9] A. Kimmig, B. Gutmann, and V. Santos Costa. Trading memory for answers: Towards tabling ProbLog. International Workshop on Statistical Relational Learning. 2009.
- [10] J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.
- [11] R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.
- [12] N. Pelov, M. Denecker, and M. Bruynooghe. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming*, 7(3):301–353, 2007.
- [13] D. Poole. Abducing through negation as failure: Stable models within the independent choice logic. *Journal of Logic Programming*, 44(1–3):5–35, 2000.
- [14] D. Poole, and N.L. Zhang. Exploiting contextual independence in probabilistic inference. *Journal of Artificial Intelligence Research*, 18:263–313, 2003.
- [15] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming*, pages 715–729. MIT Press, 1995.
- [16] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *Proceedings of the 20th International Conference on Logic Programming*. Lecture Notes in Computer Science, vol. 3132, pages 431–445. Springer, 2004.
- [17] 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*, 9(3):245–308, 2009.

A Correctness of the Mapping from LBNs to ICL

In Section 3 we introduced a mapping from LBNs to ICL. Below we show the correctness of this mapping, i.e., we show that each LBN is mapped to an ICL theory that is indeed equivalent to the LBN. Recall from Section 3.1 that we call an ICL theory (R, \mathcal{A}) *equivalent* to an LBN L if

for each input interpretation I that is legal for L it holds that the probability distribution P_{LBN} of L for I is equal to the probability distribution P_{ICL} of the ICL theory $(R \cup I, \mathcal{A})$.

Proof Sketch: First, for each LBN L and each legal input interpretation I for L it holds that the obtained ICL theory $(R \cup I, \mathcal{A})$ is **acyclic**, or at least ‘contingently acyclic’ [13, p.30]. This holds because the dependency relation (the so-called ‘parent relation’ [5]) specified by an LBN for any legal input interpretation is acyclic, and the ICL clauses in R directly capture this dependency relation (there is a direct correspondence between the targets of the probability trees in the LBN and the heads of the clauses in R , and between the parameterized random variables tested in the internal nodes of the trees in the LBN and the literals in the bodies of the clauses in R). That the ICL theory $(R \cup I, \mathcal{A})$ is acyclic implies that its semantics is well-defined, i.e., that it defines a proper probability distribution on possible worlds. We now have to prove that this distribution is equal to the distribution defined by the LBN L for the input interpretation I . Concretely, we need to prove two points: 1) to each possible world of the LBN correspond one possible world of the ICL theory and vice versa, and 2) the probability of a possible world according to the LBN is the same as the probability of the corresponding possible world of the ICL theory.

Part1: We need to prove that there is a one-to-one correspondence between the **possible worlds** of the LBN and the possible worlds of the equivalent ICL theory. Recall that a possible world of an LBN for an input interpretation I is a joint state of all random variables (RVs) in $\mathcal{RV}(I)$.

First, let us show that there is a one-to-one correspondence between the possible worlds of an LBN and the total choices of the equivalent ICL theory. Concretely, given a possible world ω_{LBN} of an LBN, the corresponding total choice is obtained as follows. Consider a particular $RV \in \mathcal{RV}(I)$. There is exactly one path in the probability tree for that RV that is ‘active’ in the world ω_{LBN} (the path that is followed for the RV in that world). With this path in the tree corresponds exactly one (grounded) clause and annotated alternative in the ICL theory (this is how we created the clauses and alternatives, recall Section 3.2.2.1). From all ground base atoms in that ground annotated alternative there is exactly one which corresponds to the observed value of the RV in the world ω_{LBN} (e.g. if the RV is *graduates(s1)* and its value in ω_{LBN} is *yes*, then the corresponding ground base atom is $b_i(s1, yes)$ where i indicates which path in the tree is active). For each $RV \in \mathcal{RV}(I)$ we can find a corresponding ground base atom in this way. The set of all ground base atoms obtained in this way constitute a total choice. This is the total choice that corresponds to the possible world ω_{LBN} .

The above shows that there is a one-to-one correspondence between the possible worlds of an LBN and the total choices of the equivalent ICL theory. The ICL semantics implies a correspondence between the total choices and the possible worlds of an ICL theory (Section 2.1). Hence, we now have a correspondence between the possible worlds of an LBN and the possible worlds of the equivalent ICL theory. From the above it should be clear that a possible world ω_{LBN} of the LBN and its corresponding possible world ω_{ICL} of the ICL theory are indeed equivalent: an RV takes a particular value (e.g. *graduates(s1)* is *yes*) in ω_{LBN} if and only if the corresponding atom (e.g. *graduates(s1, yes)*) is true in ω_{ICL} .

Part2: Above we established a one-to-one correspondence between the possible worlds of the LBN and of the equivalent ICL theory. We now prove that the **probability** of a possible world according to the LBN is the same as the probability of the corresponding possible world of the ICL theory.

The probability $P_{LBN}(\omega_{LBN})$ of a possible world ω_{LBN} according to an LBN is the product over all $RVs \in \mathcal{RV}(I)$ of the probability (according to the relevant probability tree) that the considered RV has its observed value given the observed state of its parents in ω_{LBN} . On the other hand, the probability $P_{ICL}(\omega_{ICL})$ of a possible world ω_{ICL} according the equivalent ICL theory is defined as the probability of the corresponding total choice, which is the product of the probabilities of all the ground base atoms that constitute this total choice. As shown above, to the observation that an RV takes a particular value in a possible world ω_{LBN} for the LBN corresponds one ground base atom in the equivalent ICL theory. The probability of this ground

base atom in the ICL theory is equal to the conditional probability (according to the relevant probability tree in the LBN) that the RV takes that value given its parents (this equality holds because the probabilities in the annotated alternatives in the ICL theory were chosen to be the same as those in the relevant leaf of the probability tree, recall Section 3.2.2.1). Hence, for each factor in the product P_{LBN} there is a corresponding and equal factor in the product P_{ICL} . Hence $P_{LBN} = P_{ICL}$. \square