Logic Programs with Annotated Disjunctions

Joost Vennekens and **So£e Verbaeten**[*] and **Maurice Bruynooghe**
Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A
B-3001 Leuven
Belgium
{joost,sofie,maurice}@cs.kuleuven.ac.be

Abstract

Current literature offers a number of different approaches to what could generally be called "probabilistic logic programming". These are usually based on Horn clauses. Here, we introduce a new formalism, Logic Programs with Annotated Disjunctions, based on disjunctive logic programs. In this formalism, each of the disjuncts in the head of a clause is annotated with a probability. Viewing such a set of probabilistic disjunctive clauses as a probabilistic disjunction of normal logic programs allows us to derive a possible world semantics, more precisely, a probability distribution on the set of all Herbrand interpretations. We demonstrate the strength of this formalism by some examples and compare it to related work.

Introduction

The study of the rules which govern human thought has, apart from traditional logics, also given rise to logics of probability (Halpern 2003). As was the case with £rst order logic and logic programming, attempts have been made to derive more "practical" formalisms from these probabilistic logics. Research in this £eld of "probabilistic logic programming" has mostly focused on ways in which probabilistic elements can be added to Horn clause programs. We, however, introduce in this work a formalism which is based on disjunctive logic programming (Lobo, Minker, & Rajasekar 1992).

This is natural choice, as disjunctions themselves — and therefore disjunctive logic programs — already represent a kind of uncertainty. Indeed, they can, to give just one example, be used to model indeterminate effects of actions. Consider for instance the following disjunctive clause:

$$heads(Coin) \vee tails(Coin) \leftarrow toss(Coin).$$

This clause offers quite an intuitive representation of the fact that tossing a coin will result in either heads or tails. Of course, this is not all we know. Indeed, if a coin is not biased, we know that it has equal probability of landing on heads or tails. In the formalism of *Logic Programs with Annotated Disjunctions* or *LPADs*, this can be expressed by annotating the disjuncts in the head of such a clause with a probability, i.e.

$$(heads(Coin) : 0.5) \vee (tails(Coin) : 0.5) \leftarrow toss(Coin), \neg biased(Coin).$$

Such a clause expresses the fact that for each coin c, precisely one of the following clauses will hold: $heads(c) \leftarrow toss(c), \neg biased(c)$, i.e. tossing the unbiased coin c will cause it to land on heads, or $tails(c) \leftarrow toss(c), \neg biased(c)$, i.e. tossing c will cause it to land on tails. Both these clauses have a probability of 0.5.

Such annotated disjunctive clauses can be combined to model more complicated situations. Consider for instance the following LPAD:

$$\begin{aligned}&(heads(Coin) : 0.5) \vee (tails(Coin) : 0.5)\\ &\qquad \leftarrow toss(Coin), \neg biased(Coin).\\ &(heads(Coin) : 0.6) \vee (tails(Coin) : 0.4)\\ &\qquad \leftarrow toss(Coin), biased(Coin).\\ &(fair(coin) : 0.9) \vee (biased(coin) : 0.1).\\ &(toss(coin) : 1).\end{aligned}$$

Similarly to the £rst clause, the second clause of the program expresses that a biased coin lands on heads with probability 0.6 and on tails with probability 0.4. The third clause says that a certain coin, *coin*, has a probability of 0.9 of being fair and a probability of 0.1 of being biased; the fourth clause says that *coin* is tossed (with probability 1).

In general, each body of a ground instantiation of an LPAD clause can be thought of as denoting a certain *cause*. The disjuncts in the head of such a clause sum up all possible *effects* of this cause. Each cause causes precisely one of its effects. The probabilistic annotation given to a disjunct, speci£es the probability of the body of that clause causing this effect. It is worth noting that such "causal probabilities" do not necessarily correspond to conditional probabilities. Indeed, as is well known in statistics, if the same effect can have multiple causes, which are not mutually exclusive, the conditional probability of observing this effect given that one of its causes was observed does not equal the probability of this cause actually causing its effect. Translated to LPADs, this means that if the same atom appears in the head of different clauses whose bodies are not mutually

[*]So£e Verbaeten is a Postdoctoral Fellow of the Fund for Scienti£c Research - Flanders (Belgium)(F.W.O. - Vlaanderen).

exclusive, the conditional probability of this atom given one of these bodies will be different from its annotation.

This causal interpretation of LPADs arises from the fact that, as mentioned previously, each ground instantiation of an annotated disjunctive clause is seen to represent a probabilistic choice between several non-disjunctive clauses. Similarly, each ground instantiation of an LPAD represents a probabilistic choice between several non-disjunctive logic programs, which are called *instances* of the LPAD. This intuition can be used to de£ne a probability distribution on the set of Herbrand interpretations of an LPAD: the probability of a certain interpretation I is the probability of all instances for which I is a model. This probability distribution de£nes the semantics of a program.

In the remainder of this paper, we will £rst introduce the formal syntax and semantics of LPADs. Then, we will illustrate this formalism by some examples, showing for instance how a Bayesian network and Hidden Markov Model can be represented. We will also give an overview of, and compare our work with, existing formalisms for probabilistic logic programming. It is shown that, while the ideas underlying LPADs and their semantics are not radically new, they offer enough advantages to constitute a useful contribution. An extended version of this paper is given in (Vennekens & Verbaeten 2003b).

Logic Programs with Annotated Disjunctions

A Logic Program with Annotated Disjunctions consists of a set of rules of the following form:

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow b_1, \ldots, b_m. \quad (1)$$

Here, the h_i and b_i are, respectively, atoms and literals of some language and the α_i are real numbers in the interval $[0, 1]$, such that $\sum_{i=1}^{n} \alpha_i = 1$. For a rule r of this form, the set $\{(h_i : \alpha_i) \mid 1 \leq i \leq n\}$ will be denoted as $head(r)$, while $body(r) = \{b_i \mid 1 \leq i \leq m\}$. If $head(r)$ contains only one element $(a : 1)$, we will simply write this element as a.

We will denote the set of all ground LPADs as $\mathcal{P_G}$.

The semantics of an LPAD is de£ned using its grounding. For the remainder of this section, we therefore restrict our attention to ground LPADs. Furthermore, in providing a formal semantics for such a program $P \in \mathcal{P_G}$, we will, in keeping with logic programming tradition (Lloyd 1987), also restrict our attention to its Herbrand base $H_B(P)$ and consequently to the set of all its Herbrand interpretations $\mathcal{I}_P = 2^{H_B(P)}$. In keeping with (Halpern 1989), the semantics of an LPAD will be de£ned by a probability distribution on $\mathcal{I}_P$:

De£nition 1. Let P be in $\mathcal{P_G}$. An *admissible probability distribution* π on $\mathcal{I}_P$ is a mapping from $\mathcal{I}_P$ to real numbers in $[0, 1]$, such that $\sum_{I \in \mathcal{I}_P} \pi(I) = 1$.

We would now like to select one of these admissible probability distributions as our intended semantics. To illustrate this process, we consider the grounding of the example presented in the introduction:

$$\begin{aligned}
&(heads(coin) : 0.5) \vee (tails(coin) : 0.5) \\
&\qquad\qquad \leftarrow toss(coin), \neg biased(coin). \\
&(heads(coin) : 0.6) \vee (tails(coin) : 0.4) \\
&\qquad\qquad \leftarrow toss(coin), biased(coin). \\
&(fair(coin) : 0.9) \vee (biased(coin) : 0.1). \\
&toss(coin).
\end{aligned}$$

As already mentioned in the introduction, each of these ground clauses represents a probabilistic choice between a number of non-disjunctive clauses. By choosing one of the possibilities for each clause, we get a non-disjunctive logic program, for instance:

$$\begin{aligned}
&heads(coin) \leftarrow toss(coin), \neg biased(coin). \\
&heads(coin) \leftarrow toss(coin), biased(coin). \\
&\ fair(coin). \\
&\ toss(coin).
\end{aligned}$$

Such a program is called an *instance* of the LPAD. Note that this LPAD has $2.2.2 = 8$ different instances. An instance can be assigned a probability by assuming independence between the different choices. Intuitively, this means that for each two clauses c_1, c_2 and atoms a_1, a_2 in the head of, respectively, c_1 and c_2, the probability that $body(c_1)$ causes a_1 is assumed to be independent of the probability that $body(c_2)$ causes a_2, i.e. as in (Pearl 2000), each clause is supposed to describe a single independent "causal mechanism". This assumption allows each clause to be read independently from the others, as is also the case in classical logic programming, since dependencies are modeled within one clause. In our example it of course makes perfect sense to assume that the probability of a non-biased coin landing on heads is independent of the probability of a biased coin landing on heads and of the probability of a certain coin being fair. As such, the probability of the above instance of the example is $0.5 \cdot 0.6 \cdot 0.9 \cdot 1 = 0.27$.

We now formalize the above ideas.

De£nition 2. Let P be a program in $\mathcal{P_G}$. A *selection* σ is a function which selects one pair $(h : \alpha)$ from each rule of P, i.e. $\sigma : P \rightarrow (H_B(P) \times [0, 1])$ such that for each r in P, $\sigma(r) \in head(r)$. For each rule r, we denote the atom h selected from this rule by $\sigma_{atom}(r)$ and the selected probability α by $\sigma_{prob}(r)$. Furthermore, we denote the set of all selections σ by $\mathcal{S}_P$.

Each selection σ de£nes an instance of the LPAD.

De£nition 3. Let P be a program in $\mathcal{P_G}$ and σ a selection in $\mathcal{S}_P$. The *instance* P_σ chosen by σ is obtained by keeping only the atom selected for r in the head of each rule $r \in P$, i.e. $P_\sigma = \{\text{“}\sigma_{atom}(r) \leftarrow body(r)\text{”} \mid r \in P\}$.

The process of de£ning the semantics of an LPAD through its instances, is similar to how so-called *split programs* are used in (Sakama 1990) to de£ne the *possible model semantics* for (non-probabilistic) disjunctive logic programs. The main difference is that a split program is allowed to contain more than one non-disjunctive clause for each original disjunctive clause, as the possible model semantics aims to capture both the exclusive and inclusive interpretations of disjunction. In contrast, a probabilistic rule in an LPAD expresses the fact that exactly one atom in the head holds (with a certain probability) as a consequence of the body of the rule being true[1]. It is worth noting that the semantics of the preferential reasoning formalism *Logic Programs with Ordered Disjunctions* (Brewka 2002), is also de£ned using a similar notion of instances[2].

Next, we assign a probability to each selection σ in $\mathcal{S}_P$, which induces a probability on the corresponding program P_σ. As motivated above, we assume independence between the selections made for each rule.

De£nition 4. Let P be a program in $\mathcal{P}_\mathcal{G}$. The *probability of a selection* σ in $\mathcal{S}_P$ is the product of the probabilities of the individual choices made by that selection, i.e.

$$C_\sigma = \prod_{r \in P} \sigma_{prob}(r).$$

The instances of an LPAD are normal logic programs. The meaning of such programs is given by their models under a certain formal semantics. For example, all common semantics for logic programs agree that the meaning of the above instance of the coin-program, is given by the Herbrand interpretation $\{fair(coin), toss(coin), heads(coin)\}$. The instances of an LPAD therefore de£ne a probability distribution on the set of interpretations of the program. More precisely, the probability of a certain interpretation I is the probability of all instances for which I is a model.

Returning to the example, there is one other instance of this LPAD which has $\{fair(coin), toss(coin), heads(coin)\}$ as its model, namely

$$\begin{aligned} heads(coin) &\leftarrow toss(coin), \neg biased(coin). \\ tails(coin) &\leftarrow toss(coin), biased(coin). \\ &fair(coin). \\ &toss(coin). \end{aligned}$$

The probability of this instance is $0.5 \cdot 0.4 \cdot 0.9 \cdot 1 = 0.18$. Therefore the probability of the interpretation $\{fair(coin), toss(coin), heads(coin)\}$ is $0.5 \cdot 0.4 \cdot 0.9 \cdot 1 + 0.5 \cdot 0.6 \cdot 0.9 \cdot 1 = 0.5 \cdot (0.4 + 0.6) \cdot 0.9 \cdot 1 = 0.45$.

Of course, there are a number of ways in which the semantics of a non-disjunctive logic program can be de£ned. In our framework, uncertainty is modeled by annotated disjunctions. Therefore, a non-disjunctive program should contain *no* uncertainty, i.e. it should have a single two-valued model. Indeed, this is the only way in which an LPAD can be seen as specifying a *unique* probability distribution, without assuming that the "user" meant to say something he did not actually write. Consider for instance the program: $\{a \leftarrow \neg b.\ b \leftarrow \neg a.\}$. Any reasonable probability distribution speci£ed by this program, would have to assign a probability α to the interpretation $\{a\}$ and $1-\alpha$ to $\{b\}$. However, if such a probability distribution were intended, one would simply have written: $(a : \alpha) \vee (b : 1 - \alpha)$.

Therefore, we will take the meaning of an instance P_σ of an LPAD to be given by its well founded model $WFM(P_\sigma)$ and require that all these well founded models are two-valued. If, for instance, the LPAD is acyclic (meaning that all its instances are acyclic (Apt & Bezem 1991)), this will always be the case.

De£nition 5. An LPAD P is called *sound* iff for each selection σ in $\mathcal{S}_P$, the well founded model of the program P_σ chosen by σ is two-valued.

The probabilities on the elements σ of $\mathcal{S}_P$ are then naturally extended to probabilities on interpretations. The following distribution π^*_P gives the semantics of an LPAD P.

De£nition 6. Let P be a sound LPAD in $\mathcal{P}_\mathcal{G}$. For each of its interpretations I in $\mathcal{I}_P$, the *probability* $\pi^*_P(I)$ *assigned by* P *to* I is the sum of the probabilities of all selections which lead to I, i.e. with $S(I)$ being the set of all selections σ for which $WFM(P_\sigma) = I$:

$$\pi^*_P(I) = \sum_{\sigma \in S(I)} C_\sigma.$$

It is easy to show that — for a sound LPAD — this distribution π^*_P is indeed an admissible probability distribution (Vennekens & Verbaeten 2003b).

There is a strong connection between the interpretations $I \in \mathcal{I}_P$ for which $\pi^*_P(I) > 0$ and traditional semantics for disjunctive logic programs. First of all, each such interpretation I with $\pi^*_P(I) > 0$ is also a *possible model* (Sakama 1990) of the LPAD (when ignoring the probabilities, of course). Secondly, each *stable model* (Gelfond & Lifschitz 1991) of the LPAD is such an interpretation. Moreover, in most cases, the stable model semantics coincides precisely with this set of interpretations. Only for programs in which the same atom appears in the head of different clauses, can there be a difference. Indeed, for instance the program $\{(a : 0.5) \vee (b : 0.5).\ \ a.\}$ has a unique stable model $\{a\}$, but in our probabilistic framework $\pi^*_P(\{a, b\}) = 0.5$. From a modeling perspective, this difference makes sense, because in an LPAD such a clause represents a kind of "experiment", of which the disjuncts in its head are possible outcomes. As such, there is no reason why a being true should preclude b as a possible outcome of the experiment denoted by the £rst clause.

Of course, we are not only interested in the probabilities of interpretations, but also in the probability of a formula ϕ under the semantics π^*_P. This is de£ned as the sum of the probabilities of the interpretations in which the formula is true:

De£nition 7. Let P be a sound LPAD in $\mathcal{P}_\mathcal{G}$. Slightly abusing notation, for each formula ϕ, the probability $\pi^*_P(\phi)$ of ϕ according to P is the sum of the probabilities of all inter-

[1]Of course, in such a semantics, the inclusive interpretation of disjunctions can be simulated by adding additional atoms, which explicitly represent the conjunction of two or more of the original disjuncts.

[2]We would like to thank an anonymous reviewer of a previous draft for pointing this out to us.

pretations in which ϕ holds, i.e.

$$\pi_P^*(\phi) = \sum_{I \in \mathcal{I}_P^\phi} \pi_P^*(I).$$

with $\mathcal{I}_P^\phi = \{I \in \mathcal{I}_P \mid I \models \phi\}$.

Calculating such probabilities is the basic inference task of probabilistic logic programs. Usually, the formulas are restricted to being *queries*, i.e. existentially quanti£ed conjunctions. While inference algorithms are not the focus of this work, we will nevertheless brie¤y explain how this inference task is related to inference for logic programs.

The probability of a formula is de£ned as the sum of the probabilities of all interpretations in which it is true. The probability of such an interpretation is, in turn, de£ned in terms of the probabilities of the normal logic programs which can be constructed from the LPAD. Hence, £nding a proof for the query, gives us already "part" of the probability of the query. To compute the entire probability of the query, it suf£ces to £nd all proofs and to appropriately combine the probabilities associated with the heads of the clauses appearing in these proofs.

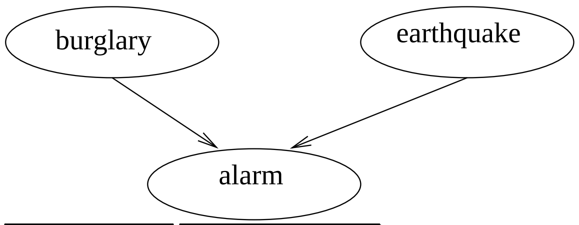
In the section on related work, we will discuss a formalism called the Independent Choice Logic (Poole 1997). For this formalism, an inference algorithm has been developed, which operates according to the principles outlined in the previous paragraph. Furthermore, a source-to-source transformation from acyclic LPADs to ICL exists, which allows this algorithm to also be applied to acyclic LPADs. Moreover, as this transformation is polynomial in the size of the input program, this shows that acyclic LPADs are in the same complexity class as ICL.

In (Vennekens & Verbaeten 2003b), we show that the semantics presented in this section is consistent with that proposed in Halpern's fundamental article (Halpern 1989), in which a general way of formalizing a certain type of probabilistic knowledge through a possible world semantics was introduced.

Examples

A Bayesian network The Bayesian network in Figure 1 can also be represented in our formalism. This is done by explicitly enumerating the possible values for each node. In this way, every Bayesian network can be represented as an LPAD.

$$\begin{aligned}
&(burg(X,t):0.1) \vee (burg(X,f):0.9).\\
&(earthq(X,t):0.2) \vee (earthq(X,f):0.8).\\
&alarm(X,t) \leftarrow burg(X,t), earthq(X,t).\\
&(alarm(X,t):0.8) \vee (alarm(X,f):0.2)\\
&\qquad\qquad \leftarrow burg(X,t), earthq(X,f).\\
&(alarm(X,t):0.8) \vee (alarm(X,f):0.2)\\
&\qquad\qquad \leftarrow burg(X,f), earthq(X,t).\\
&(alarm(X,t):0.1) \vee (alarm(X,f):0.9)\\
&\qquad\qquad \leftarrow burg(X,f), earthq(X,f).
\end{aligned}$$



burg=t	0.1
burg=f	0.9

earthq = t	0.2
earthq = f	0.8

	b=t, e=t	b=t, e=f	b=f, e=t	b=f, e=f
alarm=t	1	0.8	0.8	0.1
alarm=f	0	0.2	0.2	0.9

Figure 1: A Bayesian network.

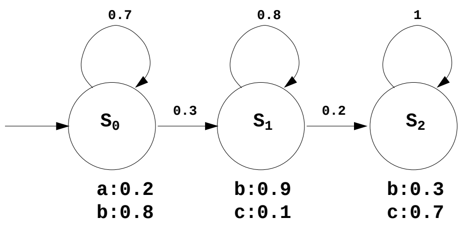


Figure 2: A Hidden Markov Model.

Actually, this LPAD represents several "versions" of the original Bayesian network, namely one for each instantiation of X. As such, this representation is similar to the *knowledge based model construction*-formalism of Bayesian Logic Programs(Kersting & Raedt 2000), a £rst-order extension of Bayesian networks, which will be discussed in the next section.

Note also that in the grounding of this LPAD, all bodies of clauses which have the same atom in their head are mutually exclusive (i.e. there is no interpretation I for which $\pi^*(I) > 0$ and two such bodies hold in I). As such, in this case, the "causal probabilities" of the LPAD can safely be interpreted as conditional probabilities.

A Hidden Markov Model The Hidden Markov Model in Figure 2 can be modeled by the following LPAD.

$$\begin{aligned}
&(state(s0,s(T)):0.7) \vee (state(s1,s(T)):0.3)\\
&\qquad\qquad \leftarrow state(s0,T).\\
&(state(s1,s(T)):0.8) \vee (state(s2,s(T)):0.2)\\
&\qquad\qquad \leftarrow state(s1,T).\\
&state(s2,s(T)) \leftarrow state(s2,T).\\
&(out(a,T):0.2) \vee (out(b,T):0.8) \leftarrow state(s0,T).\\
&(out(b,T):0.9) \vee (out(c,T):0.1) \leftarrow state(s1,T).\\
&(out(b,T):0.3) \vee (out(c,T):0.7) \leftarrow state(s2,T).\\
&state(s0,0).
\end{aligned}$$

This program corresponds nicely to the way in which one would tend to explain the semantics of this HMM in natural language. For instance, the £rst clause could be read as: "if the HMM is in state s_0, then it can either go to state s_1 or stay in state s_0." It is worth noting that this LPAD has an in£nite grounding. As such, each particular instance of this LPAD has a probability of zero. However, our semantics remains well-de£ned and still assigns an appropriate non-zero probability to each £nite string, through an in£nite sum of such zero probabilities. Moreover, the aforementioned transformation from LPADs to ICL is also able to deal with such programs and, as the inference-algorithm of ICL does not need to compute the grounding of a program, still allows these probabilities to be effectively computed as well.

Throwing dice There are some board games which require a player to roll a six (using a standard die) before he is allowed to actually start the game itself. The following example shows an LPAD which de£nes a probability distribution on how long it could take a player to do this.

$$
\begin{aligned}
&(on(D,1,s(T)) : 1/6) \vee \cdots \vee (on(D,6,s(T)) : 1/6)\\
&\qquad\qquad \leftarrow time(T), die(D), \neg on(D,6,T).\\
&start\_game(s(T)) \leftarrow time(T), on(D,6,T).\\
&time(s(T)) \leftarrow time(T).\\
&time(0).\\
&die(die).
\end{aligned}
$$

The £rst rule of this LPAD is the most important one. It states that if the player has not succeeded in getting a six on his current attempt, he will have to try again. Note that, because of the use of negation-as-failure in the body of this clause, the atoms $on(D,1,s(T)),\ldots,on(D,5,s(T))$ are only needed to serve as alternatives for $on(D,6,s(T))$. As such, in the context of this example, this clause could equivalently be written as for instance:

$$
\begin{aligned}
&(on(D,6,s(T)) : 1/6) \vee (not\_six : 5/6)\\
&\qquad\qquad \leftarrow time(T), die(D), \neg on(D,6,T).
\end{aligned}
$$

Moreover, instead of the atom not_six, any atom not appearing in the rest of the program could be used. We can therefore simply abbreviate such clauses by

$$(on(D,6,s(T)) : 1/6) \leftarrow time(T), die(D), \neg on(D,6,T).$$

Related work

There is a large body of work concerning probabilistic logic programming. Due to space limitations, we refer to (Vennekens & Verbaeten 2003a) and (Vennekens & Verbaeten 2003b) for more details.

An important class of probabilistic logic programming formalisms are those following the *Knowledge Based Model Construction* or KBMC approach. Such formalisms allow the representation of an entire "class" of propositional models, from which, for a speci£c query, an appropriate model can then be constructed "at run-time". This approach was initiated by Breese et al (Breese, Goldman, & Wellman 1994) and Bacchus (Bacchus 1993). Examples are: *Context-Sensitive Probabilistic Knowledge Bases* of Ngo and Haddawy (Ngo & Haddawy 1997), *Probabilistic Relational Models* of Getoor et al (Getoor *et al.* 2001), and *Bayesian Logic Programs* of Kersting and De Raedt (Kersting & Raedt 2000).

A formal comparison between LPADs and Bayesian Logic Programs (BLPs) is given in (Vennekens & Verbaeten 2003b). A BLP can be seen as representing a (possibly in£nite) set of Bayesian networks. Each ground atom represents a random variable, which can take on a value from a domain associated with its predicate. An implication in a clause of a BLP is not a logical implication, but rather an expression concerning probabilistic dependencies. This makes the reading of a BLP — at least for those acquainted with logic programming — less natural. Another difference is that, although it is possible to simulate classical negation in BLPs, they do not incorporate non-monotonic negation. In some cases, this can lead to longer and less intuitive programs. In (Vennekens & Verbaeten 2003b) it is formally shown that the semantics of a BLP can be expressed by an LPAD, which explicitizes the implicit argument of each atom, i.e. its "value", and enumerates all the elements in the domain. This process is similar to that which was used in the previous section to model a Bayesian network by an LPAD. Conversely, it is shown that quite a large subset of all LPADs can be represented as BLP.

Another class of formalisms, besides that of KBMC, are those which grew out of an attempt to extend logic programming with probability. Among these formalisms, *Programming in Statistical Modeling* (PRISM) (Sato & Kameya 1997; 2001) and the *Independent Choice Logic* (ICL) (Poole 1997) deviate the least from classical logic programming. ICL is a probabilistic extension of abductive logic programming. An ICL program consists of both a logical and a probabilistic part. The logical part is an acyclic, normal logic program. The probabilistic part consists of a set of clauses of the form (in LPAD syntax): $(a_1 : \alpha_1) \vee \cdots \vee (a_n : \alpha_n)$. The atoms a_i in such clauses are called abducibles. Each abducible may only appear once in the probabilistic part of an ICL program; in the logical part of the program, abducibles may only appear in the bodies of clauses.

Syntactically, each ICL program is clearly an LPAD. In (Vennekens & Verbaeten 2003b) it was shown that this embedding of ICL into LPADs preserves the original semantics of ICL (as formulated in (Poole 1997)). Conversely, each acyclic LPAD can be transformed into one in this restricted syntax (Vennekens & Verbaeten 2003b). This is done by creating new, arti£cial atoms, which explicitly represent the process of choosing a disjunct from the head of a clause, as is illustrated by the following ICL-version of the coin-example of the introduction:

$$\begin{array}{l} heads(Coin) \\ \quad \leftarrow toss(Coin), \neg biased(Coin), fair\_heads(Coin). \\ tails(Coin) \\ \quad \leftarrow toss(Coin), \neg biased(Coin), fair\_tails(Coin). \\ heads(Coin) \\ \quad \leftarrow toss(Coin), biased(Coin), biased\_heads(Coin). \\ tails(Coin) \\ \quad \leftarrow toss(Coin), biased(Coin), biased\_tails(Coin). \\ (fair\_heads(Coin) : 0.5) \vee (fair\_tails(Coin) : 0.5). \\ (biased\_heads(Coin) : 0.6) \vee (biased\_tails(Coin) : 0.4). \\ (fair(coin) : 0.9) \vee (biased(coin) : 0.1). \\ (toss(coin) : 1). \end{array}$$

On the £rst author's web site[3] a Prolog program can be found which performs this transformation. As such, even though LPADs do not (yet) have an implemented inference algorithm of their own, it is already possible to solve queries to acyclic LPADs by using the ICL algorithm.

It should be noted that, although these two formalisms are similar in terms of theoretical expressive power, they are quite different in their approach to modeling uncertainty. Indeed, ICL (and of course the corresponding subset of LPADs) is ideally suited for problem domains such as diagnosis or theory revision, in which it is most natural to express uncertainty on the *causes* of certain effects. The extended expressiveness of LPADs (in the sense that LPADs allow more natural representations of certain types of knowledge) on the other hand, makes these also suited for problems such as modeling indeterminate actions, in which it is most natural to express uncertainty on the *effects* of certain causes. Of course, this is not surprising, as a similar relationship exists between the non-probabilistic formalisms on which ICL and LPADs are based: (Sakama & Inoue 1994) shows that abductive logic programs and disjunctive logic programs are essentially equivalent; however, history has shown that both these formalisms are valid ways of representing knowledge, with each having problem domains for which it is better suited than the other.

LPADs are not the only probabilistic formalism based on disjunctive logic programming. In *Many-Valued Disjunctive Logic Programs* of Lukasiewicz (Lukasiewicz 2001) probabilities are associated with disjunctive clauses as a whole. In this way, uncertainty of the implication itself — and not, as is the case with LPADs, of the disjuncts in the head — is expressed.

All the works mentioned above use point probabilities. There are however also a number of formalisms using probability intervals: *Probabilistic Logic Programs* of Ng and Subrahmanian (Ng & Subrahmanian 1992), their extension to *Hybrid Probabilistic Programs* of Dekhtyar and Subrahmanian (Dekhtyar & Subrahmanian 2000) and *Probabilistic Deductive Databases* of Lakshmanan and Sadri (Lakshmanan & Sadri 1994). Contrary to our approach, programs in these formalisms do not de£ne a $single$ probability distribution, but rather a *set* of possible probability distributions, which — in a sense — allows one to express a kind of "meta-uncertainty", i.e. uncertainty about which distribution is the "right" one. Moreover, the techniques used by these formalisms tend to have more in common with constraint logic programming than "normal" logic programming.

[3]http://www.cs.kuleuven.ac.be/∼joost.

Conclusion and future work

We have introduced the formalism of Logic Programs with Annotated Disjunctions. In our opinion, this formalism offers a natural and consistent way of describing complex probabilistic knowledge in terms of a number of (independent) simple choices, an idea which is prevalent in for instance (Poole 1997). Furthermore, it does not ignore the crucial concept of conditional probability, which underlies the entire "Bayesian movement", and does not deviate from the well established and well known non-probabilistic semantics of £rst-order logic and logic programming. Indeed, for an LPAD P, the set of interpretations I for which $\pi^*_P(I) > 0$, is a subset of the possible models of P and a (small) superset of its stable models.

While the comparison with related work such as ICL showed that this is not a radically new approach, we feel its additional expressiveness (in the sense that LPADs allow more natural representations of certain types of knowledge) offers enough advantages to constitute a useful contribution to the £eld of probabilistic logic programming. In future work, we hope to demonstrate this further, by presenting larger, real-world applications of LPADs. We also plan further research concerning a proof procedure and complexity analysis for LPADs. Finally, there are a number of possible extensions to the LPAD formalism which should be investigated. For example, it might prove useful to allow (classical) negation in the head of LPAD rules or to incorporate aggregates in order to allow a more concise representation of certain basic probability distributions. Because of the logical nature of LPADs and their instance-based semantics, it should be fairly straightforward to add such extensions to the language in a natural way.

References

Apt, K., and Bezem, M. 1991. Acyclic programs. *New Generation Computing* 9:335–363.

Bacchus, F. 1993. Using £rst-order probability logic for the construction of bayesian networks. In *Proceedings of the Sixth Conference on Uncertainty in Arti£cial Intelligence,* 219–226.

Breese, J.; Goldman, R.; and Wellman, M. 1994. Introduction to the special section on knowledge-based construction of probabilistic and decision models. *IEEE Transactions on Systems, Man, and Cybernetics* 24(11):1577–1579.

Brewka, G. 2002. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Arti£cial Intelligence, AAAI-2002. Morgan Kaufmann, 2002.,* 100–105.

Dekhtyar, A., and Subrahmanian, V. 2000. Hybrid probabilistic programs. *Journal of Logic Programming* 43(3):187–250.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New generation computing* 9:365–385.

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

Halpern, J. 1989. An analysis of £rst-order logics of probability. *Arti£cial Intelligence* 46:311–350.

Halpern, J. Y. 2003. *Reasoning about uncertainty*. MIT press.

Kersting, K., and Raedt, L. D. 2000. Bayesian logic programs. In Cussens, J., and Frisch, A., eds., *Work-in-Progress Reports of the Tenth International Conference on Inductive Logic Programming (ILP-2000)*.

Lakshmanan, L., and Sadri, F. 1994. Probabilistic deductive databases. In Bruynooghe, M., ed., *Proceedings of the International Symposium on Logic Programming*, 254–268. MIT Press.

Lloyd, J. 1987. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition.

Lobo, J.; Minker, J.; and Rajasekar, A. 1992. *Foundations of Disjunctive Logic Programming*. MIT Press.

Lukasiewicz, T. 2001. Fixpoint characterizations for many-valued disjunctive logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, volume 2173 of *Lecture Notes in Arti£cial Intelligence*, 336–350. Springer-Verlag.

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

Ngo, L., and Haddawy, P. 1997. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science* 171(1–2):147–177.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

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

Sakama, C., and Inoue, K. 1994. On the equivalence between disjunctive and abductive logic programs. In *International Conference on Logic Programming*, 489–503.

Sakama, C. 1990. Possible model semantics for disjunctive databases II (extended abstract). In *Logic Programming and Non-monotonic Reasoning*, 107–114.

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

Sato, T., and Kameya, Y. 2001. Statistical abduction with tabulation. *Kowalski 60th birthday volume*.

Vennekens, J., and Verbaeten, S. 2003a. A general view on probabilistic logic programming. In *Proceedings of the 15th Belgian-Dutch Conference on Arti£cial Intelligence*, 299–306. http://www.cs.kuleuven.ac.be/∼joost/bnaic.ps.

Vennekens, J., and Verbaeten, S. 2003b. Logic programs with annotated disjunctions. Technical Report CW386, K.U. Leuven. http://www.cs.kuleuven.ac.be/∼joost/techrep.ps.