

Logic Programs with Annotated Disjunctions

Joost Vennekens Sofie Verbaeten

Report CW 368, September 2003



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

Logic Programs with Annotated Disjunctions

Joost Vennekens Sofie Verbaeten

Report CW 368, September 2003

Department of Computer Science, K.U.Leuven

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.

Keywords : uncertainty reasoning, logic programming

1 Introduction

The study of the rules which govern human thought has, apart from traditional logics, also given rise to logics of probability [14]. As was the case with first order logic and logic programming, attempts have been made to derive more “practical” formalisms from these probabilistic logics. Research in this field 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 [22].

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. the unbiased coin c will land on heads when tossed, or $tails(c) \leftarrow toss(c), \neg biased(c)$, i.e. the unbiased coin c will land on tails when tossed. 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:

Example 1.

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

Similarly to the first 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 certainly (with probability 1) tossed.

As mentioned previously, each ground instantiation of an annotated disjunctive clause represents 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 define 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. As in the work of Halpern [15], this probability distribution defines the semantics of a program.

In the next section, we give a very brief overview of the work of Halpern [15]. In that work, a distinction is made between two different types of semantics of logics of probability. Our formalism, as well as most other formalisms for probabilistic logic programming, has a type 2 semantics. This second type of semantics is explained in more detail in Section 2. Next, we describe our formalism in Section 3. We formally define its syntax (Section 3.1) and semantics (Section 3.2). We also formalize the relation between our semantics and the type 2 semantics as proposed by Halpern (Section 3.3). Following the approach of Halpern, we define the probability of a formula in Section 3.4. We illustrate our formalism further by presenting some examples in Section 4. It is shown how a Hidden Markov Model and Bayesian network can be represented by an LPAD, and how LPADs can represent actions with uncertain effects in a situation calculus setting. In Section 5 we give an overview of, and compare our work with, existing formalisms for probabilistic logic programming. In particular, we classify the different approaches into three categories, and formally compare LPADs with a single formalism from each category, namely with (Section 5.1) the Independent Choice Logic of Poole [28], with (Section 5.2) Bayesian Logic Programs of Kersting and De Raedt [19, 17], and with (Section 5.3) Stochastic Logic Programs of Muggleton and Cussens [8, 24]. It is shown that, while the semantics of LPADs is similar to that of some existing approaches, they do offer significant advantages by providing a natural way of representing relational probabilistic knowledge and as such constitute a useful contribution to the field of probabilistic logic programming. We conclude and discuss future work in

Section 6.

2 Fundamentals

In [15], Halpern makes some important fundamental observations concerning first order logics of probability. In particular, he distinguishes two different types of semantics for such a logic. The first approach, so-called type 1 semantics, allows statements about the probability that some property holds for a *random* element taken from a certain domain. The second approach, type 2 semantics, allows one to speak about the probability of a property holding for a *particular* element from a domain. Traditionally, the intuition behind this distinction is made clear by presenting the two following sentences.

- Type 1: “The probability that a randomly chosen bird flies, is greater than 0.9.”
- Type 2: “The probability that Tweety (a particular bird) flies, is greater than 0.9.”

As can be observed from these statements, in a type 1 semantics one knows with certainty whether each particular bird (e.g. Tweety) flies and “randomness” is generated by the process of choosing a bird from a set of birds. In a type 2 semantics, randomness follows from it both being possible that a particular bird does fly and it being possible that this same bird does not fly.

Most existing probabilistic logic programming formalisms follow a type 2 approach (see Section 5), as will our formalism (see Section 3). So, for our purposes, it suffices to summarize how a type 2 semantics can be formalized.

Consider some alphabet \mathcal{A} . The language $\mathcal{L}_2(\mathcal{A})$ contains the formulae consisting of logical connectives and both atoms constructed from \mathcal{A} in the usual way and atoms of the form $w(\phi)$. Here, ϕ is an arbitrary formula and w is a special function, intuitively meaning “the probability of ϕ ”. For instance, the above sentence can be formalized as “ $w(\textit{flies}(\textit{tweety})) \geq 0.9$ ”.

A type 2 semantics can then be described by a tuple $M = (D, S, \mu, \pi)$, where D is a domain, S is a set of *states* or *possible worlds*, μ assigns to each $s \in S$ an interpretation of the function and predicate symbols in \mathcal{A} over D and π is a probability distribution over S . The interpretation of classical

formulae by the tuple M in a certain state s is defined in the usual way. The interpretation of $w(\phi)$ (technically once again in a state s) is defined as $\pi(\{s' \in S \mid (M, s') \models \phi\})$, i.e. the probability of the states in which ϕ holds. Finally, one can then say that $M \models \phi$, for any formula $\phi \in \mathcal{L}_2(\mathcal{A})$, if and only if $(M, s) \models \phi$ for every state $s \in S$.

3 Logic Programs with Annotated Disjunctions

We formally define the syntax (Section 3.1) and semantics (Section 3.2) of LPADs. In Section 3.3 we describe the relationship between our semantics and the type 2 semantics as proposed by Halpern. By following the approach of Halpern, we define the probability of a formula in Section 3.4.

3.1 Syntax

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

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

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 .

3.2 Semantics

The semantics of an LPAD is defined using its grounding. For the remainder of this section, we therefore restrict our attention to ground LPADs. We denote the set of all ground LPADs as \mathcal{P}_G . Furthermore, in providing a formal semantics for such a program $P \in \mathcal{P}_G$, we will, in keeping with logic programming tradition [21], 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 [15], the semantics of an LPAD will be defined by a probability distribution on \mathcal{I}_P .

Definition 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:

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

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:

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

Such a program is called an *instance* of the LPAD. Note that this LPAD has $2 \cdot 2 \cdot 2 = 8$ different instances. Such an instance can be assigned a probability by assuming independence between the different choices. This is a reasonable assumption to make, because — as in classical logic programming — it should be possible to read each clause independently from the others; dependence should be modeled within one clause. Indeed, in our example it 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 these ideas. The notion of a selection function formalizes the idea of choosing, for each ground instantiation of a rule in an LPAD, between the atoms in its head:

Definition 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 , $\sigma(r) \in \text{head}(r)$. 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 defines an instance of the LPAD.

Definition 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. $P_\sigma = \{\sigma_{atom}(r) \leftarrow \text{body}(r) \mid r \in P\}$.

The process of defining the semantics of an LPAD through its instances, is similar to how so-called *split programs* are used in [29] to define 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. Of course, in such a semantics, the inclusive interpretation of disjunctions can be simulated by adding additional atoms to explicitly represent the conjunction of two or more of the original disjuncts. It is worth noting that the semantics of the preferential reasoning formalism *Logic Programs with Ordered Disjunctions* [6], is also defined using a similar notion of instances¹.

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.

Definition 4. Let P be a program in \mathcal{P}_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

¹We would like to thank an anonymous reviewer of an older version of this report for pointing this out to us.

of the above instance of the coin-program, is given by the Herbrand interpretation $\{toss(coin), fair(coin), heads(coin)\}$. The instances of an LPAD therefore define 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 $\{toss(coin), fair(coin), heads(coin)\}$ 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 $\{toss(coin), fair(coin), heads(coin)\}$ 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 defined. 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 specified 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 [2]), this will always be the case.

Definition 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 .

Definition 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.

Proposition 1. *Let P be a sound LPAD in $\mathcal{P}_{\mathcal{G}}$. Then π_P^* is an admissible probability distribution.*

Proof. Let P be a sound, ground LPAD. Then clearly $\sum_{\sigma \in \mathcal{S}_P} C_\sigma = 1$. Furthermore, because P is sound, for each σ in \mathcal{S}_P the well founded model of P_σ is in \mathcal{I}_P . Therefore π_P^* is an admissible probability distribution. \square

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* [29] of the LPAD (when ignoring the probabilities, of course). Secondly, each *stable model* [12] 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 where the same atom appears in the head of different clauses, as for instance in $\{(a : 0.5) \vee (b : 0.5). a.\}$, can there be a difference. Indeed, this example 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 heads 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 first clause.

3.3 Relation with Halpern’s work

In Halpern’s terminology [15] as presented in Section 2, restricting to Herbrand interpretations corresponds to taking $D = H_B(P)$, $S = \mathcal{I}_P$ and μ the trivial function $\mu_{=}$ assigning function symbols and predicate symbols to themselves. Each probability distribution on \mathcal{I}_P corresponds to such a type 2 interpretation of the program.

Following Halpern's approach, we can derive a \models -relation, by stating that a rule $r \in P$ of form (1) corresponds to the following set $Halp(r)$ of formulae:

$$\begin{aligned} h_1 \vee \dots \vee h_n &\leftarrow b_1 \wedge \dots \wedge b_m. \\ w(h_1 \wedge b_1 \wedge \dots \wedge b_m) &\geq \alpha_1 \cdot w(b_1 \wedge \dots \wedge b_m). \\ &\vdots \\ w(h_n \wedge b_1 \wedge \dots \wedge b_m) &\geq \alpha_n \cdot w(b_1 \wedge \dots \wedge b_m). \end{aligned}$$

Now, taking the set of interpretations with a non-zero probability as the set of possible states, we can simply define:

Definition 7. For each $P \in \mathcal{P}_G$, $\pi \in \Pi_P$, $r \in P$:

$$\pi \models_{Halpern} r \quad \text{iff} \quad (H_B(P), \pi^{-1}([0, 1]), \mu_{=}, \pi) \models Halp(r).$$

Our intended semantics π_P^* can be shown to be consistent with this $\models_{Halpern}$ -relation. To do this, we need the following proposition.

Proposition 2. Let P be a sound LPAD in \mathcal{P}_G . Let $\sigma \in \mathcal{S}_P$ and r a clause in P such that $WFM(P_\sigma) \models body(r)$. Let $\sigma' \in \mathcal{S}_P$ such that for each clause $r' \neq r$ in P , $\sigma'(r') = \sigma(r')$. Then $WFM(P_{\sigma'}) \models body(r)$.

Proof. If $\sigma'(r) = \sigma(r)$, $P_\sigma = P_{\sigma'}$ and the proof is trivial. So suppose $\sigma(r) = (h_i : \alpha_i)$ and $\sigma'(r) = (h_j : \alpha_j)$, with $i \neq j$. Since $WFM(P_\sigma) \models body(r)$, there is a justification J (see [11]) of $body(r)$ in P_σ with value *true*. We now prove that this justification J is also a justification of $body(r)$ in $P_{\sigma'}$. We first prove that the rule $h_i \leftarrow body(r)$ is not used in the justification J . Suppose this rule would be used. Then, since $WFM(P_\sigma) \models body(r)$, it would be used to prove that h_i is true. So, in the justification of $body(r)$, there would be a positive loop (which has truth value *false*) and this is impossible. Finally, in proving that J is also a justification in $P_{\sigma'}$, we prove that J does not contain a proof that h_j is false. Suppose there is such a proof in J , then, in transforming J into a justification in $P_{\sigma'}$, we would have to add a proof that $body(r)$ is false. But this would give us, in the proof of $body(r)$ in $P_{\sigma'}$ a proof of $\neg body(r)$, and hence a loop over negation (which has truth value *undefined*). Since the LPAD P is sound, it can not be the case that $P_{\sigma'}$ has a three-valued well-founded model, therefore, in J there is no proof that h_j is false. This concludes the proof. \square

Theorem 1. For each $P \in \mathcal{P}_G$:

$$\pi_P^* \models_{Halpern} P.$$

Proof. Let $r = (h_1 : \alpha_1) \vee \dots \vee (h_n : \alpha_n) \leftarrow b_1 \wedge \dots \wedge b_m$ be a rule in P and let I be in \mathcal{I}_P , such that $\pi_P^*(I) > 0$. Because I is the well founded model of P_σ for some selection $\sigma \in \mathcal{S}_P$, $I \models h_1 \vee \dots \vee h_n \leftarrow b_1 \wedge \dots \wedge b_m$.

Secondly, for each $(h_i : \alpha_i)$ in the head of r and each $\sigma \in \mathcal{S}_P$ such that $\sigma_{atom}(r) = h_i$ and $WFM(P_\sigma) \models body(r)$, it clearly holds that $WFM(P_\sigma) \models h_i$. Therefore:

$$\pi_P^*(h_i \wedge body(r)) = \sum_{\substack{\sigma \in \mathcal{S}_P \\ WFM(P_\sigma) \models h_i \wedge body(r)}} C_\sigma \geq \sum_{\substack{\sigma \in \mathcal{S}_P \\ WFM(P_\sigma) \models body(r) \\ \sigma(r) = h_i}} C_\sigma$$

Now, let σ be an element of \mathcal{S}_P such that $WFM(P_\sigma) \models body(r)$. Because of proposition 2, for each $\sigma' \in \mathcal{S}_P$, such that for all $r' \neq r$ $\sigma'(r') = \sigma(r')$, $WFM(P_{\sigma'}) \models body(r)$. Therefore, let $(\sigma_j)_{j \in J}$, be all the selections in \mathcal{S}_P such that $WFM(P_{\sigma_j}) \models body(r)$ and $(\sigma_j)_{atom} = h_i$. Then we have that the set $\{\bar{\sigma}_j \mid j \in J\}$ is a partition of the set $\{\sigma \in \mathcal{S}_P \mid WFM(P_\sigma) \models body(r)\}$, where $\sigma \in \bar{\sigma}_j$ iff for each clause $r' \neq r$: $\sigma(r') = \sigma_j(r')$. Then:

$$\begin{aligned} \pi_P^*(body(r)) &= \sum_{\substack{\sigma \in \mathcal{S}_P \\ WFM(P_\sigma) \models body(r)}} C_\sigma = \sum_{j \in J} \prod_{r' \neq r} (\sigma_j)_{prob}(r') \\ &= \frac{1}{\alpha_i} \sum_{j \in J} C_{\sigma_j} = \frac{1}{\alpha_i} \sum_{\substack{\sigma \in \mathcal{S}_P \\ WFM(P_\sigma) \models body(r) \\ \sigma(r) = h_i}} C_\sigma. \end{aligned}$$

So, we have proven that $\pi_P^*(h_i \wedge body(r)) \geq \alpha_i \pi_P^*(body(r))$. \square

3.4 Probability of a formula

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^* . Following Halpern's work, the probability of a formula ϕ under the semantics π_P^* is given by the sum of the probabilities of the interpretations in which the formula is true:

Definition 8. Let P be a sound LPAD in \mathcal{P}_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 interpretations 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 quantified conjunctions. While inference algorithms are not the focus of this work, we will nevertheless briefly explain how this inference task is related to inference for logic programs.

The probability of a formula is defined as the sum of the probabilities of all interpretations in which it is true. The probability of such an interpretation is, in turn, defined in terms of the probabilities of the normal logic programs which can be constructed from the LPAD. Hence, finding a proof for the query, gives us already “part” of the probability of the query. To compute the entire probability of the query, it suffices to find all proofs and to appropriately combine the probabilities associated with the heads of the clauses appearing in these proofs.

In Section 5 on related work, we will discuss a formalism called the Independent Choice Logic [28]. 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.

4 Examples

In this section, we present some examples, to demonstrate the strength of the LPAD formalism.

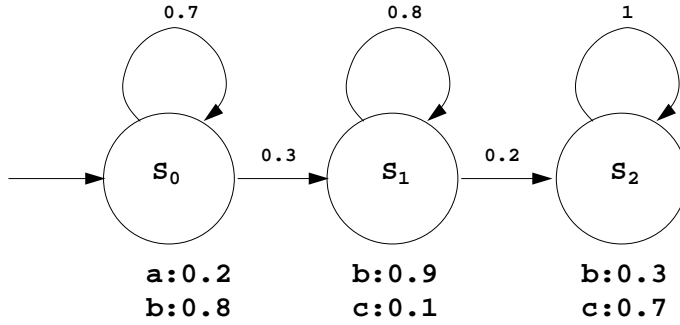


Figure 1: A Hidden Markov Model.

4.1 A Hidden Markov Model

The Hidden Markov Model in Figure 1 can be modeled by the following LPAD.

$$\begin{aligned}
 & (state(s_0, T + 1) : 0.7) \vee (state(s_1, T + 1) : 0.3) \leftarrow state(s_0, T). \\
 & (state(s_1, T + 1) : 0.8) \vee (state(s_2, T + 1) : 0.2) \leftarrow state(s_1, T). \\
 & \quad \quad \quad state(s_2, T + 1) \leftarrow state(s_2, T). \\
 & (out(a, T) : 0.2) \vee (out(b, T) : 0.8) \leftarrow state(s_0, T). \\
 & (out(b, T) : 0.9) \vee (out(c, T) : 0.1) \leftarrow state(s_1, T). \\
 & (out(b, T) : 0.3) \vee (out(c, T) : 0.7) \leftarrow state(s_2, T). \\
 & \quad \quad \quad state(s_0, 0).
 \end{aligned}$$

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 first 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 infinite grounding. As such, each particular instance of this LPAD has a probability of zero. However, our semantics remains well-defined and still assigns an appropriate non-zero probability to each finite string, through an infinite sum of such zero probabilities. Moreover, the aforementioned transformation from LPADs to ICL is also able to deal with such programs, since it does not instantiate any variables. As the inference-algorithm of ICL does not need to compute the

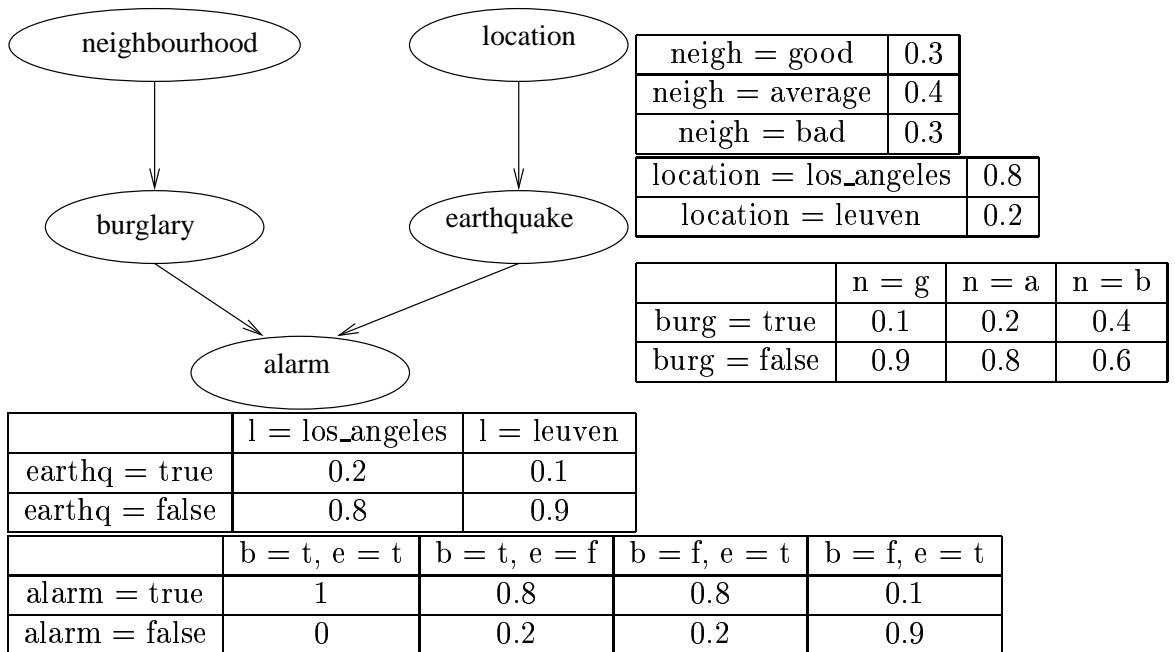


Figure 2: A Bayesian network.

grounding of a program, but rather searches for “proofs” of a query in an SLD-like manner, this allows these probabilities to be effectively computed as well.

4.2 A Bayesian network

In this section, we will show how the Bayesian network in Figure 2 can be represented in our formalism. This is done by explicitly enumerating the possible values for each node. Every Bayesian network can be represented as an LPAD in this way.

$$\begin{aligned}
& (\text{neigh}(X, \text{good}) : 0.3) \vee (\text{neigh}(X, \text{average}) : 0.4) \vee (\text{neigh}(X, \text{bad}) : 0.3). \\
& \quad (\text{location}(X, \text{los_angeles}) : 0.8) \vee (\text{location}(X, \text{leuven}) : 0.2). \\
& \quad (\text{burg}(X, \text{true}) : 0.1) \vee (\text{burg}(X, \text{false}) : 0.9) \leftarrow \text{neigh}(X, \text{good}). \\
& (\text{burg}(X, \text{true}) : 0.2) \vee (\text{burg}(X, \text{false}) : 0.8) \leftarrow \text{neigh}(X, \text{average}). \\
& \quad (\text{burg}(X, \text{true}) : 0.4) \vee (\text{burg}(X, \text{false}) : 0.6) \leftarrow \text{neigh}(X, \text{bad}). \\
& \quad (\text{earthq}(X, \text{true}) : 0.2) \vee (\text{earthq}(X, \text{false}) : 0.8) \\
& \quad \quad \leftarrow \text{location}(X, \text{los_angeles}). \\
& \quad (\text{earthq}(X, \text{true}) : 0.1) \vee (\text{earthq}(X, \text{false}) : 0.9) \\
& \quad \quad \leftarrow \text{location}(X, \text{leuven}). \\
& \text{alarm}(X, \text{true}) \leftarrow \text{burg}(X, \text{true}), \text{earthq}(X, \text{true}). \\
& \quad (\text{alarm}(X, \text{true}) : 0.8) \vee (\text{alarm}(X, \text{false}) : 0.2) \\
& \quad \quad \leftarrow \text{burg}(X, \text{true}), \text{earthq}(X, \text{false}). \\
& \quad (\text{alarm}(X, \text{true}) : 0.8) \vee (\text{alarm}(X, \text{false}) : 0.2) \\
& \quad \quad \leftarrow \text{burg}(X, \text{false}), \text{earthq}(X, \text{true}). \\
& \quad (\text{alarm}(X, \text{true}) : 0.1) \vee (\text{alarm}(X, \text{false}) : 0.9) \\
& \quad \quad \leftarrow \text{burg}(X, \text{false}), \text{earthq}(X, \text{false}).
\end{aligned}$$

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 [18], a first-order extension of Bayesian networks, which is discussed in Section 5. Furthermore, our semantics allows the easy incorporation of known facts, such as $\text{location}(\text{joost}, \text{leuven})$.

4.3 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 defines a probability distribution on how long it could take a player to do this.

$$\begin{aligned}
& (on(D, 1, s(T)) : 1/6) \vee (on(D, 2, s(T)) : 1/6) \vee \dots \vee (on(D, 6, s(T)) : 1/6) \\
& \quad \leftarrow time(T), die(D), \neg on(D, 6, T). \\
& start_game(s(T)) \leftarrow time(T), on(D, 6, T). \\
& \quad time(s(T)) \leftarrow time(T). \\
& \quad \quad time(0). \quad die(die).
\end{aligned}$$

The first 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 $on(D, 1, s(T)), \dots, on(D, 5, s(T))$ 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:

$$(on(D, 6, s(T)) : 1/6) \vee (not_six : 5/6) \leftarrow time(T), die(D), \neg on(D, 6, T).$$

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

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

4.4 Flipping coins

Elaborating slightly on the coin-example from the introduction, this section shows a situation calculus-like representation of repeated experiments with two different types of coins. On the one hand, there are *repetitive* coins, which when thrown tend to repeat their previous position. More precisely, if the previous position was heads, such a coin will land on heads again with probability 75%; if the previous position was tails, the probability of heads is only 25%. Expressing this by two LPAD rules (one for each previous position) is straightforward. On the other hand, *fair* coins simply land on heads or tails with equal probability. In the LPAD, the fact that the new position of such coins is independent of their previous position, is clear because the body of the clause representing the “fair coin”-experiment does not refer to the previous position. Finally, if a coin is not known to be repetitive, we assume it is fair.

$$\begin{aligned}
& (\text{shows}(C, \text{heads}, \text{throw}(C, S)) : 0.75) \vee (\text{shows}(C, \text{tails}, \text{throw}(C, S)) : 0.25) \\
& \quad \leftarrow \text{coin}(C), \text{repetitive}(C), \text{state}(S), \text{shows}(C, \text{heads}, S). \\
& (\text{shows}(C, \text{tails}, \text{throw}(C, S)) : 0.75) \vee (\text{shows}(C, \text{heads}, \text{throw}(C, S)) : 0.25) \\
& \quad \leftarrow \text{coin}(C), \text{repetitive}(C), \text{state}(S), \text{shows}(C, \text{tails}, S). \\
& (\text{shows}(C, \text{heads}, \text{throw}(C, S)) : 0.5) \vee (\text{shows}(C, \text{tails}, \text{throw}(C, S)) : 0.5) \\
& \quad \leftarrow \text{coin}(C), \neg \text{repetitive}(C), \text{state}(S). \\
& \text{shows}(C1, F, \text{throw}(C2, S)) \\
& \quad \leftarrow \text{shows}(C1, F, S), \text{coin}(C1), \text{coin}(C2), C1 \neq C2, \text{state}(S). \\
& \quad \text{shows}(\text{bad_coin}, \text{heads}, s0). \quad \text{coin}(\text{bad_coin}). \\
& \quad \text{shows}(\text{good_coin}, \text{heads}, s0). \quad \text{coin}(\text{good_coin}). \\
& \quad \text{repetitive}(\text{bad_coin}). \quad \text{state}(s0). \\
& \text{state}(\text{throw}(\text{Coin}, S)) \leftarrow \text{coin}(\text{Coin}), \text{state}(S).
\end{aligned}$$

This LPAD defines a probability distribution on the position of each of the coins after they have been thrown a number of times. For instance $\pi_P^*(\text{shows}(\text{bad_coin}, \text{heads}, \text{throw}(\text{good_coin}, \text{throw}(\text{bad_coin}, s0)))) = 0.75$ is the probability of the repetitive coin showing *heads*, after first throwing the repetitive coin and then the fair coin.

4.5 Shooting turkeys

We consider the following variant of the turkey shooting problem. If we try to shoot a healthy turkey, there are three possible outcomes: we can miss (with probability 0.2), we can hit but merely wound it (with probability 0.5) or we can immediately succeed in killing it (with probability 0.3). Trying to shoot a wounded turkey can have two possible effects: we can miss, in which case the turkey simply remains wounded (with probability 0.3 — wounded turkeys are of course less mobile than their healthy counterparts), or we can hit and kill it (with probability 0.7). If the turkey is already wounded and we wish to save our bullets, we can also simply wait to see whether it will succumb to its wounds (with probability 0.4). However, this also gives the turkey a chance to rest and as such it could recover from its injuries (with probability 0.1).

The following LPAD models this problem, using the situation calculus.

$$\begin{aligned}
& (\text{holds}(\text{healthy}, \text{do}(\text{shoot}, S)) : 0.2) \vee \text{holds}(\text{wounded}, \text{do}(\text{shoot}, S)) : 0.5) \\
& \quad \vee \text{holds}(\text{dead}, \text{do}(\text{shoot}, S)) : 0.3) \leftarrow \text{holds}(\text{healthy}, S). \\
& (\text{holds}(\text{wounded}, \text{do}(\text{shoot}, S)) : 0.3) \vee (\text{holds}(\text{dead}, \text{do}(\text{shoot}, S)) : 0.7) \\
& \quad \leftarrow \text{holds}(\text{wounded}, S). \\
& (\text{holds}(\text{healthy}, \text{do}(\text{wait}, S)) : 0.1) \vee (\text{holds}(\text{wounded}, \text{do}(\text{wait}, S)) : 0.5) \\
& \quad \vee \text{holds}(\text{dead}, \text{do}(\text{shoot}, S)) : 0.4) \leftarrow \text{holds}(\text{wounded}, S). \\
& \text{holds}(\text{dead}, \text{do}(A, S)) \leftarrow \text{holds}(\text{dead}, S), \text{action}(A). \\
& \text{holds}(\text{Prop}, \text{do}(\text{wait}, S)) \leftarrow \text{holds}(\text{Prop}, S), \neg \text{holds}(\text{wounded}, S). \\
& \text{holds}(\text{healthy}, s_0). \quad \text{action}(\text{wait}). \quad \text{action}(\text{shoot}).
\end{aligned}$$

The probability distribution defined by this LPAD specifies, for instance, that the probability $\pi_P^*(\text{holds}(\text{dead}, \text{do}(\text{wait}, \text{do}(\text{shoot}, s_0))))$ of the turkey being dead after shooting and waiting, is $0.3 + 0.5 \cdot 0.4 = 0.5$. Once again, this probability can also be effectively computed by applying the transformation from LPADs to ICL.

5 Related work

There is a large body of work concerning probabilistic logic programming. Since discussing every one of these formalisms would lead us too far, we will first classify them into three groups and then content ourselves with comparing LPADs to a single formalism from each category. Finally, we will summarize our conclusions in Section 5.4.

Probabilistic logic programming formalisms can first of all be categorized by whether they attempt to express *Halpern type 1* knowledge or *type 2* knowledge². Most formalisms express Halpern type 2 knowledge. A notable exception is *Stochastic Logic Programs* of Muggleton and Cussens [25], which we'll discuss in Section 5.3.

The formalisms expressing Halpern type 2 knowledge can be divided further according to whether the formalisms grew out of an attempt to extend logic programming with probability or one to construct a first-order version of a well-known propositional probabilistic framework. These last formalisms

²While there exist approaches to combine these two types of knowledge, all of these fall outside the scope of probabilistic logic programming.

allow the representation of an entire “class” of propositional models, from which, for a specific query, an appropriate model can then be constructed “at run-time”. This approach, usually referred to as *Knowledge Based Model Construction* (KMBC), was initiated by Breese et al [5] and Bacchus [3]. Examples are: *Bayesian Logic Programs* of Kersting and De Raedt [18] (see Section 5.2), *Context-Sensitive Probabilistic Knowledge Bases* of Ngo and Haddawy [27], and *Probabilistic Relational Models* of Getoor et al [13] (based on relational databases).

Among the formalisms extending logic programming with probability, the *Independent Choice Logic* [28] (see Section 5.1), and *Programming in Statistical Modeling* (PRISM) [31, 32] deviate the least from classical logic programming. They associate probabilities with certain atoms (hypotheses), while the rules are kept classical.

LPADs are not the only probabilistic formalism based on disjunctive logic programming. In *Many-Valued Disjunctive Logic Programs* of Lukasiewicz [23] 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. In our work, the goal is not to represent uncertainty about the truth of a disjunctive clause. In fact, given an LPAD and a model of it, all the corresponding non-probabilistic disjunctive rules are true in the interpretations which have non-zero probability. Instead, LPADs are to be used in situations where one has uncertainty about the consequence (head) of a given conjunction of atoms (body).

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 [26], their extension to *Hybrid Probabilistic Programs* of Dekhtyar and Subrahmanian [10] and *Probabilistic Deductive Databases* of Lakshmanan and Sadri [20]. Contrary to our approach, programs in these formalisms do not define 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 probability distribution is the “right” one. Moreover, the techniques used by these formalisms have, in general, more in common with constraint logic programming than “normal” logic programming.

In Section 5.1, LPADs will be compared to the Independent Choice Logic, which, like LPADs, is a type 2 extension of logic programming. In Section 5.2 a comparison will be made with Bayesian Logic Programming, which follows the KBMC-approach. [17] contains a comparison between Bayesian Logic

Programming and other KBMC-approaches. Finally, in Section 5.3 we will discuss the relationship with Stochastic Logic Programs, which represent type 1 knowledge. [7] contains a comparison between Stochastic Logic Programs and several other formalisms.

In the following sections, we will explain the syntax and semantics of the formalism under consideration, provide a formal comparison between its expressive power and that of LPADs and discuss the (dis-)advantages of the various different ways of representing probabilistic knowledge.

5.1 Independent Choice Logic

5.1.1 Syntax and semantics

An Independent Choice Logic [28] program consists of two parts:

- A set C of declarations of the following form:

$$\mathit{random}([a_1 : \alpha_1, \dots, a_n : \alpha_n]),$$

with a_i atoms, $\alpha_i \in [0, 1]$, such that $\sum \alpha_i = 1$.

- A normal, acyclic logic program F .

Furthermore, such a pair (C, F) must satisfy the following conditions: no atom a_i appearing in a *random*-statement in C may unify with another atom a_j which also appears in a *random*-statement in C and no head of a clause in F may unify with an atom a_i appearing in C .

A total choice χ is a selection of one atom a_{ij} from each ground instantiation j of the *random*-declarations in C . The set of all total choices for an ICL program P , is denoted by \mathbf{C}_P . Each total choice $\chi \in \mathbf{C}_P$ has a probability C_χ , which is the product $\prod_j \alpha_{ij}$ of the probabilities α_{ij} of the atoms a_{ij} chosen from the ground *random*-declarations j . Furthermore, each total choice χ “explains” one interpretation $I \in \mathcal{I}_P$, namely that which is the unique stable model³ of the new program P_χ , consisting of F augmented with the chosen facts a_{ij} . The semantics of an ICL program P is the probability distribution π_P^{ICL} , which assigns C_χ to an interpretation explained by χ and zero to the interpretations for which no explanation exists.

³Each acyclic logic program has a unique stable model [4].

Example 2 gives the ICL version of the coin example (Example 1) of the introduction. Note that the process of choosing a disjunct from the head of a clause is done by creating new, artificial atoms, such as e.g. $fair_heads(Coin)$.

Example 2. ICL version of Example 1:

$$\begin{aligned}
&heads(Coin) \leftarrow toss(Coin), \neg biased(Coin), fair_heads(Coin). \\
&tails(Coin) \leftarrow toss(Coin), \neg biased(Coin), fair_tails(Coin). \\
&heads(Coin) \leftarrow toss(Coin), biased(Coin), biased_heads(Coin). \\
&tails(Coin) \leftarrow toss(Coin), biased(Coin), biased_tails(Coin). \\
&toss(coin). \\
&random([fair_heads(Coin) : 0.5, fair_tails(Coin) : 0.5]). \\
&random([biased_heads(Coin) : 0.6, biased_tails(Coin) : 0.4]). \\
&random([fair(coin) : 0.9, biased(coin) : 0.1]).
\end{aligned}$$

5.1.2 Formal comparison

It is easy to see that there is a syntactic 1-1 correspondence between the set of all ICL-programs and a subset \mathcal{P}_{ICL} of all LPADs. To formalize this, we introduce the following subprograms of an LPAD P :

$$\begin{aligned}
P_C &= \{r \in P \mid body(r) = \{\}, |head(r)| > 1\} \\
P_F &= \{r \in P \mid |head(r)| = 1\}
\end{aligned}$$

We can now define \mathcal{P}_{ICL} as the set of all LPADs P which satisfy the following conditions:

- C1. P is acyclic.
- C2. $P = P_F \cup P_C$.
- C3. No atom in P_C unifies with an atom in the head of a clause of P_F or a different atom in P_C .

Clearly, the transformation α which maps each rule $(h_1 : \alpha_1) \vee \dots \vee (h_n : \alpha_n)$ in P_C to the atom $random[(h_1 : \alpha_1), \dots, (h_n : \alpha_n)]$ and leaves all other rules unchanged, is a bijection between \mathcal{P}_{ICL} and the set \mathcal{ICL} of all ICL programs. Furthermore, α also respects the semantics, i.e.

Theorem 2. *Let P be an LPAD in \mathcal{P}_{ICL} . Then $\pi_P^* = \pi_{\alpha(P)}^{ICL}$.*

Proof. Because we only consider acyclic programs, we do not need to distinguish between the well founded model and the unique stable model [4]. Therefore, it suffices to prove that there exists a bijection k from \mathcal{S}_P to the set $\mathbf{C}_{\alpha(P)}$ of total choices, which maps each σ in \mathcal{S}_P to a total choice χ , such that $P_\sigma = P_\chi$ and $C_\sigma = C_\chi$. So let σ be some element of \mathcal{S}_P and let $k(\sigma)$ be the total choice χ which chooses from each *random*-statement r the atom $\sigma_{atom}(r)$. Then k is clearly bijective. Furthermore, $C_\chi = \prod_{r \in P} \sigma_{prob}(r) = C_\sigma$ and $P_\chi = P_F \cup \bigcup_{r \in P} \sigma_{atom}(r) = P_\sigma$. \square

However, it turns out that each acyclic LPAD can be transformed to a program in \mathcal{P}_{ICL} . Intuitively, this is done by “flattening” the program, i.e. creating new artificial atoms (cfr. *fair_heads(Coin)* in Example 2) to explicitly represent the process of choosing a disjunct from the head of a clause. To allow for a simpler notation, we will only define this transformation for ground LPADs. The definition can, however, easily be extended to the non-ground case.

For a ground instantiation r of some LPAD clause, which has the following form:

$$(h_1 : \alpha_1) \vee \dots \vee (h_n : \alpha_n) \leftarrow b_1, \dots, b_m.$$

with the h_i atoms and the b_i literals, we define $\beta(r)$ as being the following set of clauses:

$$\begin{aligned} h_1 &\leftarrow b_1, \dots, b_m, choice_r(1). \\ &\vdots \\ h_n &\leftarrow b_1, \dots, b_m, choice_r(n). \\ (choice_r(1) : \alpha_1) &\vee \dots \vee (choice_r(n) : \alpha_n). \end{aligned}$$

We define the transformation $\beta(P)$ of a ground LPAD P as the union $\bigcup_{r \in P} \beta(r)$. It is clear that if P is acyclic, then $\beta(P)$ is in \mathcal{P}_{ICL} . Furthermore, the transformation β preserves the semantics of the program.

Theorem 3. *Let P be a ground LPAD. Then the restriction of $\pi_{\beta(P)}^*$ to the atoms of P equals π_P^* .*

Proof. Once again, it suffices to prove that there exists a bijection k between \mathcal{S}_P and $\mathcal{S}_{\beta(P)}$, such that for each σ in \mathcal{S}_P , $WFM(P_\sigma) = WFM(P_{k(\sigma)})$ (when

restricted to the atoms in P) and $C_\sigma = C_{k(\sigma)}$. For each rule $r \in P$ of the above form and selection function σ of \mathcal{S}_P , let $s_\sigma(r)$ be the unique natural number i for which $\sigma(r) = (h_i : \alpha_i)$. For each $\sigma \in \mathcal{S}_P$, we then define $k(\sigma)$ as the function which selects from a clause $(choice_r(1) : \alpha_1) \vee \dots \vee (choice_r(n) : \alpha_n)$ in some $\beta(r)$ the pair $(choice_r(s_\sigma(r)) : \alpha_{s_\sigma(r)})$. This k is clearly bijective and furthermore $C_\sigma = C_{k(\sigma)}$.

Because all atoms $(choice_r(j) : \alpha_j)$ for which $j \neq s_\sigma(r)$ only appear positively in the bodies of the program $P_{k(\sigma)}$ and not in the heads, we can eliminate each clause containing such an atom from $P_{k(\sigma)}$, without changing its well founded model. Furthermore, because in the remaining program, each atom $choice_r(j)$ is also a fact, we can also remove these atoms from the program altogether and still have the same well founded model. This leaves precisely P_σ . \square

The combination of these two theorems gives us both a transformation α^{-1} from ICL programs to (acyclic) LPADs and a transformation $\alpha \circ \beta$ from acyclic LPADs to ICL programs.

Therefore, the only LPADs which cannot be transformed to an ICL-program are those for which there exists a $\sigma \in \mathcal{S}_P$ such that the well founded model of P_σ is two-valued, but P_σ is not acyclic. Example 3 contains an example of such a program and its semantics. One would, however, not expect to encounter such programs in practice.

Example 3.

$$P = \{(a : 0.5) \vee (b : 0.5). \\ a \leftarrow b. \\ b \leftarrow a.\}$$

$$\pi^* : \mathcal{I}_P \rightarrow [0, 1] : \begin{cases} \{a, b\} \mapsto 1 \\ I \mapsto 0 \end{cases} \quad I \neq \{a, b\}.$$

5.1.3 Conclusion

Although ICL and LPADs are similar in terms of theoretical expressive power, they are nevertheless quite different in their practical modeling properties. 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 greater 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: [30] proves that abductive logic programming and disjunctive logic programming 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.

In conclusion of this section, we would like to mention that the existence of such a simple transformation from LPADs to ICL-programs shows that we achieve the aforementioned greater expressiveness without significantly increasing the computational complexity. Indeed, a simple program performing the transformation described above (e.g. that which can be found on the first author's website⁴) suffices to provide acyclic LPADs with a fully implemented proof procedure.

5.2 Bayesian Logic Programming

5.2.1 Syntax and semantics

Bayesian Logic Programming [19, 17], or BLP for short, is based on the (propositional) concept of Bayesian networks. Each ground atom a with a predicate p represents a random variable, which can take on a value from a domain $d_p = \{v_1^p, \dots, v_{m_p}^p\}$ associated with p . A clause in a BLP is of the form

$$H \mid A_1, \dots, A_n. \tag{2}$$

with H and the A_i atoms. Each of these clauses must be range-restricted, i.e. every variable in its head must also appear in its body. Such a clause requires a conditional probability table (CPT) to be associated with it, specifying the conditional probability of each possible value for the random variables corresponding to the atom in the head, given the values of those corresponding to the atoms in the body. We will assume that, for each ground atom a ,

⁴<http://www.cs.kuleuven.ac.be/~joost>.

the set of all ground instances of the clauses of a BLP contains at most one clause with a in its head⁵. Slightly abusing the notation, the Least Herbrand Model of a BLP B ($LHM(B)$) is defined as the Least Herbrand Model of the definite logic program obtained from B by replacing the “|” symbols with “ \leftarrow ”.

The semantics of a BLP is defined by relating it back to a Bayesian network. Each element of the Least Herbrand Model of the program corresponds to a node in this network. There is an edge from a node representing ground atom X to a node representing ground atom Y if the program contains a clause $c = X' | \dots, Y', \dots$, for which a substitution θ exists, such that $X'\theta = X$ and $Y'\theta = Y$. The CPT associated with a node is simply that which was associated with the clause for the atom corresponding to this node.

In order for this semantics to work, this possibly infinite Bayesian network must obviously be well defined. In [16], it is shown to be sufficient if the program is acyclic and there are only a finite number of parent nodes for each node.

We can also make the semantics of a BLP B more explicit and say that it defines a probability distribution on the set \mathcal{V}_B of all possible ways in which each random variable, i.e. each element of the Least Herbrand Model $LHM(B)$, can be assigned a value from its domain. Formally, the set \mathcal{V}_B is defined as:

$$\mathcal{V}_B = \{ \nu \mid \nu : LHM(B) \rightarrow \bigcup_{a \in LHM(B)} dom(a) \text{ such that:} \\ \forall a \in LHM(B) : \nu(a) \in dom(a) \}$$

where we write $dom(a)$ to denote the domain associated with the predicate symbol of a . A BLP B then defines the following probability distribution π_B^{BLP} on \mathcal{V}_B :

$$\forall \nu \in \mathcal{V}_B : \pi_B^{BLP}(\nu) = \prod_{h \in LHM(B)} \alpha^h,$$

⁵Technically speaking, Bayesian Logic Programming does allow multiple ground clauses with the same atom in the head. However, in this case, a so-called “combination rule” must be given, which specifies how these clauses with their CPTs can be combined into a single clause with one CPT. We therefore do not lose generality by making this assumption.

with α^h the entry in row “ $\nu(h)$ ” and column “ $b_1 = \nu(b_1), \dots, b_n = \nu(b_n)$ ” of the CPT for the unique clause $h \mid b_1, \dots, b_n$ with h in its head.

Example 4 shows how the coin example (Example 1) of the introduction can be modeled as a BLP. The key difference between these two representations is that some arguments which are explicit in the LPAD-version, become implicit in the BLP. This clearly makes it harder (if not impossible) to “read” the BLP in the same way as one would read a logic program.

Example 4. BLP version of Example 1:

$$\begin{aligned} & \textit{side}(\textit{Coin}) \leftarrow \textit{toss}(\textit{coin}), \textit{biased}(\textit{coin}). \\ & \textit{biased}(\textit{coin}). \\ & \textit{toss}(\textit{coin}). \end{aligned}$$

	toss(coin)		biased(coin)		
	t	1	t	0.6	
	f	0	f	0.4	
side(C)	t(C) = t, b(C) = t	t(C) = t, b(C) = f	t(C) = f, b(C) = t	t(C) = f, b(C) = f	
heads	0.6	0.5	0	0	
tails	0.4	0.5	0	0	
NA	0	0	1	1	

Note that, in order to simulate the fact that we are only interested in the position of coins *after* they have been tossed, the domains of the random variables corresponding to ground instantiations of $\textit{side}(C)$ had to be extended with the “don’t care”-value *NA* (not applicable).

5.2.2 Formal comparison

Because the semantics of both BLPs and LPADs are based on ground instantiations of the program, in this section we will only consider ground BLPs and LPADs, as this makes the notations somewhat less cumbersome.

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 d_p . This process is similar to that which was used in Section 4.2 to model a Bayesian network by an LPAD.

We will transform each column c in a CPT to an LPAD clause $\gamma(c)$. So let c be some column of the following form

H	\dots	$B_1 = v_{i_1}^{B_1}, \dots, B_n = v_{i_n}^{B_n}$	\dots
v_1^H		α_1	
\vdots		\vdots	
v_n^H		α_n	

with H and the B_i atoms, each $v_j^{B_i}$ an element of the domain associated with the predicate of B_i and the set $\{v_1^H, \dots, v_n^H\}$ the domain of the predicate of H . For an atom $A = p(t_1, \dots, t_n)$ and a term t , we will denote the new atom $p(t_1, \dots, t_n, t)$ with $A(t)$. The transformation $\gamma(c)$ of a column c is then the following clause:

$$(H\langle v_1^H \rangle : \alpha_1) \vee \dots \vee (H\langle v_n^H \rangle : \alpha_n) \leftarrow B_1\langle v_{i_1}^{B_1} \rangle, \dots, B_n\langle v_{i_n}^{B_n} \rangle.$$

For each clause r in a BLP B , we denote by $\gamma(r)$ the set of clauses $\gamma(c)$ for each column c of the CPT for r and by $\gamma(B)$ the union of all $\gamma(r)$ for each $r \in B$.

Because the Bayesian network corresponding to a BLP B has to be acyclic, we can divide its nodes, i.e. the elements of $LHM(B)$, into a number of different “levels” λ_i .

$$\lambda_0 = \{a \in LHM(B) \mid a \text{ does not have parents}\},$$

$$\forall i > 0 : \lambda_i = \{a \in LHM(B) \mid i - 1 \text{ is the smallest } j \text{ for which } Par(a) \subseteq \bigcup_{k \leq j} \lambda_k\}.$$

Because of the condition that each element of $LHM(B)$ is only influenced by a finite number of other elements of $LHM(B)$, there exists a $n \in \mathbb{N}$ for which $LHM(B) = \{\lambda_0, \dots, \lambda_n\}$.

Using this partition of $LHM(B)$, we can now prove the following theorem, showing that the semantics of $\gamma(B)$ indeed matches that of the BLP B .

Theorem 4. *Let B be a ground BLP. For each $\nu \in \mathcal{V}_B$:*

$$\pi_B^{BLP}(\nu) = \pi_{\gamma(B)}^*(\{a\langle \nu(a) \rangle \mid a \in LHM(B)\}).$$

Proof. For each $\nu \in \mathcal{V}_B$, we define $k(\nu)$ as the set of all $\sigma \in \mathcal{S}_{\gamma(B)}$, which for each head $a \in LHM(B)$ of a clause “ $a \mid b_1, \dots, b_n$ ” of B , select the atom $a\langle \nu(a) \rangle$ from the clause in $\gamma(B)$ which has $b_1\langle \nu(b_1) \rangle, \dots, b_n\langle \nu(b_n) \rangle$ as its body. It is clear that $\sum_{\sigma \in k(\nu)} C_\sigma = \pi_B^{BLP}(\nu)$. Therefore it suffices to show that for each $\sigma \in k(\nu)$, $WFM(P_\sigma) = \{a\langle \nu(a) \rangle \mid a \in LHM(B)\}$. We will do this

by induction on the levels $\lambda_0, \dots, \lambda_n$ of $LHM(B)$. Let ν_i be the restriction of ν to $\cup_{j \leq i} \lambda_j$. We define $k(\nu_i)$ as the union of all sets $k(\mu)$, for which the restriction of μ to $\cup_{j \leq i} \lambda_j$ equals ν_i . We will show that for each i in $0..n$:

$$\sigma \in k(\nu_i) \quad \text{iff} \quad WFM(P_\sigma) \cap \bigcup_{j \leq i} \lambda_j = \{a \langle \nu(a) \rangle \mid a \in \bigcup_{j \leq i} \lambda_j\}$$

For each a in λ_0 , there is exactly one clause in $\gamma(B)$ with the atoms $(a \langle v \rangle)_{v \in \text{dom}(a)}$ in its head and furthermore this clause has an empty body. So clearly, each atom $a \langle v \rangle$ for $a \in \lambda_0$ will only be in $WFM(P_\sigma)$ if σ chooses this $a \langle v \rangle$ from that clause. Now, suppose the equivalence holds for all $j < i$ and let a be an atom in λ_i . Then each clause in $\gamma(B)$ with a $a \langle v \rangle$ in its head, must be in $\gamma(r)$ with r the unique rule $a \mid b_1, \dots, b_n$ in B . Furthermore, each of these b_i is in $\cup_{j < i} \lambda_j$. Therefore, by the induction hypothesis, the only clause which can cause an atom $a \langle v \rangle$ to be in $WFM(P_\sigma)$, is the clause in $\gamma(r)$ with body $b_1 \langle \nu(b_1) \rangle, \dots, b_n \langle \nu(b_n) \rangle$. Therefore an atom $a \langle v \rangle$ being in $WFM(P_\sigma)$ is equivalent to $a \langle v \rangle$ being chosen from that rule. \square

This result shows that we can consider the set of all BLPs to be a subset of all LPADs. This subset is however quite large. Indeed, we will show that each ground acyclic LPAD, such that each ground atom only depends on a finite number of atoms⁶, can be transformed to a BLP. We will denote the set of these LPADs by \mathcal{P}_{BLP} . Let P be an element of this set. For each rule $r \in P$ of the following form:

$$(h_1 : \alpha_1) \vee \dots \vee (h_n : \alpha_n) \leftarrow b_1, \dots, b_m, \neg c_1, \dots, \neg c_l.$$

we define $\delta(r)$ as the following BLP clause:

$$\text{choice}_r \mid b_1, \dots, b_m, c_1, \dots, c_l.$$

with the following CPT (all the atoms of P have domain $\{true, false\}$; the newly introduced atom choice_r has domain $\{h_1, \dots, h_n, none\}$):

choice_r	\dots	$b_1 = true, \dots, b_m = true, c_1 = false, \dots, c_l = false$	\dots
h_1	0	α_1	0
\vdots			
h_n	0	α_n	0
$none$	1	0	1

⁶An atom h *directly depends* on an atom b , if there is a clause with h in its head and b in its body. An atom h *depends* on b , if (h, b) belongs to the transitive closure of this “directly depends on”-relation.

For each atom a in $H_B(P)$, which appears in the head of clauses r_1, \dots, r_n we define $\gamma(a)$:

$$a \mid \text{choice}_{r_1}, \dots, \text{choice}_{r_n}.$$

with as CPT the table which has for each column containing $\text{choice}_{r_i} = a$ for some i a 1 in row $a = \text{true}$ and for all other columns a 1 in row $a = \text{false}$.

For each atom a in $H_B(P)$ which does not appear in the head of any clause, we define $\gamma(a)$ as the fact a with the following CPT:

a	
true	0
false	1

We then define $\beta(P)$ as the union of all $\delta(r)$ for each $r \in P$ and all $\gamma(a)$ for $a \in H_B(P)$.

For instance, the LPAD clause

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

would be transformed to the set of clauses

$$\begin{aligned} \text{heads}(\text{Coin}) &\leftarrow \text{toss}(\text{Coin}), \text{not_biased}(\text{Coin}), \text{ch}(\text{Coin}). \\ \text{tails}(\text{Coin}) &\leftarrow \text{toss}(\text{Coin}), \text{not_biased}(\text{Coin}), \text{ch}(\text{Coin}). \\ &\text{ch}(\text{Coin}). \end{aligned}$$

with the following CPTs:

$\text{heads}(C)$	$t(C) = t, \text{nb}(C) = t, \text{ch}(C) = 1$	\dots
t	1	0
f	0	1
$\text{tails}(C)$	$t(C) = t, \text{nb}(C) = t, \text{ch}(C) = 2$	\dots
t	1	0
f	0	1

$\text{ch}(C)$	
1	0.5
2	0.5

To show that this transformation indeed preserves the semantics, we need the following two propositions, in which we denote for a $\nu \in \mathcal{V}_{\beta(P)}$ the set $\{\sigma \in \mathcal{S}_P \mid \forall r \in P : \text{if } \nu(\text{choice}_r) \neq \text{none} \text{ then } \sigma_{\text{atom}}(r) = \nu(\text{choice}_r)\}$ by $S(\nu)$.

Proposition 3. *Let P be in \mathcal{P}_{BLP} . Then for each $\nu \in \mathcal{V}_{\beta(P)}$, such that $\pi_{\beta(P)}^{BLP}(\nu) > 0$:*

$$\sum_{\sigma \in S(\nu)} C_{\sigma} = \pi_{\beta(P)}^{BLP}(\nu).$$

Proof. By definition, $\pi_{\beta(P)}^{BLP}(\nu)$ equals the product of the elements α_i^r from the rows h_i from the CPTs of all rules r , for which $\nu(\text{choice}_r) \neq \text{none}$. Therefore $\pi_{\beta(P)}^{BLP}(\nu)$ is either equal to zero or to $\sum_{\sigma \in S(\nu)} C_\sigma$. \square

Proposition 4. *Let P be in \mathcal{P}_{BLP} . Then for each $\nu \in \mathcal{V}_{\beta(P)}$ such that $\pi_{\beta(P)}^{BLP}(\nu) > 0$:*

$$\forall \sigma \in S(\nu) : WFM(P_\sigma) = \{a \in H_B(P) \mid \nu(a) = \text{true}\}.$$

Proof. Let P be in \mathcal{P}_{BLP} , let ν be an element of $\mathcal{V}_{\beta(P)}$, such that $\pi_{\beta(P)}^{BLP}(\nu) > 0$ and let σ be an element of $S(\nu)$. We can split the elements of $H_B(P_\sigma)$ into a finite number of levels λ_i by defining:

$$\lambda_i = \{a \in H_B(P) \mid i \text{ is the smallest } j \text{ for which:}$$

$$\forall r \in P, \text{ such that } a \in \text{head}(r) : \forall b \in \text{body}(r) : b \in \bigcup_{l < j} \lambda_l\}$$

We will show by induction that for each i :

$$WFM(P_\sigma) \cap \bigcup_{j \leq i} \lambda_j = \{a \in H_B(P) \mid \nu(a) = \text{true}\} \cap \bigcup_{j \leq i} \lambda_j$$

An atom a in λ_0 is only in $WFM(P_\sigma)$ if P_σ contains a fact a . In this case, $\nu(a) = \text{true}$ and else, because of the clause for a , $\nu(a) = \text{false}$.

An atom a in λ_i is only in $WFM(P_\sigma)$ if P_σ contains a rule $a \leftarrow \text{Body}$, such that $WFM(P_\sigma) \models \text{Body}$. By the induction hypothesis, it must be the case that for each $b \in \text{Body}$, $b \in WFM(P_\sigma)$ iff $\nu(b) = \text{true}$. Therefore, because $\pi_{\beta(P)}^{BLP}(\nu) > 0$, $\nu(a) = \text{true}$. \square

Theorem 5. *Let P be in \mathcal{P}_{BLP} . Then for each $I \in \mathcal{I}_P$, such that $\pi_P^*(I) > 0$:*

$$\pi_P^*(I) = \sum_{\nu \in \mathcal{V}_{\beta(P)}^I} \pi_{\beta(P)}^{BLP}(\nu)$$

with $\mathcal{V}_{\beta(P)}^I = \{\nu \in \mathcal{V}_{\beta(P)} \mid \forall a \in H_B(P) : \nu(a) = \text{true} \text{ iff } a \in I\}$.

Proof. Let P be in \mathcal{P}_{BLP} and let I be an interpretation in \mathcal{I}_P . Let $V(I) = \{\nu \in \mathcal{V}_{\beta(P)}^I \mid \pi_{\beta(P)}^{BLP}(\nu) > 0\}$ and let $S(I) = \{\sigma \in \mathcal{S}_P \mid WFM(P_\sigma) = I\}$. Because of proposition 3, it suffices to show that $\{S(\nu) \mid \nu \in V(I)\}$ is a partition of $S(I)$.

Because of proposition 4, $\bigcup_{\nu \in V(I)} S(\nu) \subseteq S(I)$. To prove the inclusion in the other direction, let σ be an element of $S(I)$. Let ν be the element of \mathcal{V}_P , which maps each atom in I to *true*, each atom in $H_B(P) \setminus I$ to *false*, each *choice_r* for which $I \models \text{body}(r)$ to $\sigma_{\text{atom}}(r)$ and each other *choice_r* to *none*. Then clearly $\sigma \in S(\nu)$ and $\nu \in \mathcal{V}_{\beta(P)}^I$. Furthermore, because for each a such that $\nu(a) = \text{true}$ there exists a rule $a \leftarrow \text{Body}$ in P_σ such that $I \models \text{Body}$, $\pi_{\beta(P)}^{BLP}(\nu) > 0$.

Because for each $\nu, \mu \in V(I)$, $\{\text{choice}_r \in H_B(\beta(P)) \mid \nu(\text{choice}_r) = \text{none}\} = \{\text{choice}_r \mid I \not\models \text{body}(r)\} = \{\text{choice}_r \in H_B(\beta(P)) \mid \mu(\text{choice}_r) = \text{none}\}$, if $\nu \neq \mu$, there must be an atom *choice_r*, for which $\nu(\text{choice}_r) = a$ and $\mu(\text{choice}_r) = b$, with $a \neq b$. Therefore $S(\nu) \cap S(\mu) = \{\}$. □

5.2.3 Conclusion

It is our opinion that the construction of a Bayesian network might not be the best way to define the semantics of a probabilistic logic program. First of all, it seems to lead to — at least for those acquainted with logic programming — less intuitive programs, since a logical atom can no longer be interpreted as such, but must instead be thought of as a collection of random variables. Furthermore, it is clear from Example 4, that the “|”-sign cannot be interpreted as an implication, but must instead be thought of as an expression concerning the probabilistic dependencies between these random variables.

Secondly, the expressive power of the formalism appears to be unnecessarily restricted. While acyclicity is perhaps not an unreasonable assumption to make, this approach’s inherent lack of non-monotonic negation does appear to be a disadvantage.

5.3 Stochastic Logic Programs

A Stochastic Logic Program [8, 24], or SLP for short, consists of a set of range-restricted definite Horn clauses. Each clause c has a label $\lambda_c \in [0, 1]$, which specifies the probability that c is used in an SLD-refutation when an atom with the same predicate as the head of c needs to be resolved. As such, denoting by C_p the set of all clauses with a predicate symbol p in the head,

the condition that $\sum_{c \in C_p} \lambda_c = 1$ is imposed⁷.

SLPs represent type 1 knowledge. More precisely, the probability of each ground instantiation $p(a_1, \dots, a_n)$ of a predicate p/n is defined as the sum of the probabilities of each proof for $p(a_1, \dots, a_n)$, normalized by the sum of the probabilities of each proof for $p(X_1, \dots, X_n)$. In this formula, the probability of a proof using clause c a number of $t(c)$ times is $\prod_c \lambda_c^{t(c)}$.

Due to the correctness and completeness of SLD-resolution (for definite programs), a transformation from SLPs to LPADs exists, such that the ratios of $P_{\pi^*}(p(a_1, \dots, a_n))$ to $P_{\pi^*}(p(X_1, \dots, X_n))$ give the semantics of the SLP. This requires explicitly stating that for each clause c with predicate p in its head, a choice must be made between either deriving the head of this clause or deciding that another clause for p will be used. Also, some atoms need to be given an extra argument to differentiate between subsequent “calls” of this atom.

While [1] contains a theoretical description of how a type 2 semantics can be simulated in a type 1 setting, the peculiarities of the semantics of SLPs make it unclear whether and how certain LPADs could be transformed to SLPs. As such, we are (currently) even unable to provide a SLP-version of Example 1.

Although defining the probability of an instantiation of a predicate in terms of SLD-proofs might seem reasonable when taking into account SLP’s origin as a generalization of stochastic context-free grammars and hidden Markov models, this does lead to some (in our view) undesirable properties. For instance, in a SLP, the clause $\lambda : a \leftarrow b$ is *not* equivalent to the clause $\lambda : a \leftarrow b \wedge b$.

The tight coupling of SLP’s semantics to SLD-resolution, seems to make it practically impossible to obtain a declarative reading of a SLP. In our opinion, only having a procedural view severely limits one’s ability to write meaningful SLPs.

5.4 Summary

In general, the novelty of our approach lies in the use of annotated disjunctive clauses as “probabilistic primitives”. This distinguishes our formalism from for instance ICL, which only allows disjunctive clauses with empty bodies, from BLPs, where implicit arguments are used to avoid explicit disjunctions

⁷For simplicity, here we only consider what in [9] are called “normalized, pure SLPs”.

and from SLPs, where uncertainty is associated with the SLD-inference algorithm.

6 Conclusion and future work

In Section 3, Logic Programs with Annotated Disjunctions were introduced. 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 [28]. 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 first-order logic and logic programming. Indeed, as shown in Section 3.2, 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 (Section 5) showed that the ideas underlying this formalism and its semantics are not radically new, we feel it offers enough additional advantages in providing a natural representation of relational probabilistic knowledge, to constitute a useful contribution to the field 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 the use of variables in the probabilistic annotations and incorporate aggregates, in order to allow a more concise representation of certain basic probability distributions. In such a way, one would be able to express, for instance, that if one chooses a person at random from a room in which there are m men and f women, the probability of having chosen a man is $\frac{m}{m+f}$:

$$(male(C) : \frac{M}{P}) \vee (female(C) : \frac{F}{P}) \leftarrow chosen(C),$$

$$M = count(X, male(X)), F = count(X, female(X)), P = M + F.$$

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. In other formalisms, such as BLP or ICL, this appears to be more difficult.

References

- [1] M. Abadi and J. Y. Halpern. Decidability and expressiveness for first-order logics of probability. *Information and Computation*, 112(1):1–36, 1994).
- [2] K.R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.
- [3] F. Bacchus. Using first-order probability logic for the construction of bayesian networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 219–226, 1993.
- [4] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.
- [5] J.S. Breese, R.P. Goldman, and M.P. Wellman. 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, 1994.
- [6] G. Brewka. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI-2002. Morgan Kaufmann, 2002.*, pages 100–105, 2002.
- [7] J. Cussens. Integrating probabilistic and logical reasoning. *Electronic Transactions on Artificial Intelligence*, 1999.
- [8] J. Cussens. Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 115–122. MK, 2000.
- [9] J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.
- [10] A. Dekhtyar and V.S. Subrahmanian. Hybrid probabilistic programs. *Journal of Logic Programming*, 43(3):187–250, 2000.

- [11] M. Denecker and D. De Schreye. Justification semantics: a unifying framework for the semantics of logic programs. In L.M. Pereira and A. Nerode, editors, *Proceedings of the Second International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 365–379. MIT press, 1993.
- [12] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9:365–385, 1991.
- [13] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 7–34. Springer-Verlag, 2001. to appear.
- [14] J. Y. Halpern. *Reasoning about uncertainty*. MIT press, 2003.
- [15] J.Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1989.
- [16] K. Kersting. Bayes’sche-logische Programme. Master’s thesis, Albert-Ludwigs-University, Freiburg, Germany, 2000.
- [17] K. Kersting and L. De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.
- [18] K. Kersting and L. De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Work-in-Progress Reports of the Tenth International Conference on Inductive Logic Programming (ILP-2000)*, 2000.
- [19] K. Kersting and L. De Raedt. Bayesian Logic Programs. Technical report, Institute for Computer Science, University of Freiburg, Germany, april 2001.
- [20] L.V.S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In M. Bruynooghe, editor, *Proceedings of the International Symposium on Logic Programming*, pages 254–268. MIT Press, 1994.
- [21] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.

- [22] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.
- [23] T. Lukasiewicz. 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 Artificial Intelligence*, pages 336–350. Springer-Verlag, 2001.
- [24] S. Muggleton. Stochastic logic programs. *Journal of Logic Programming*, 2001. Accepted subject to revision.
- [25] S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.
- [26] R.T. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.
- [27] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1–2):147–177, 1997.
- [28] D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.
- [29] C. Sakama. Possible model semantics for disjunctive databases II (extended abstract). In *Logic Programming and Non-monotonic Reasoning*, pages 107–114, 1990.
- [30] C. Sakama and K. Inoue. On the equivalence between disjunctive and abductive logic programs. In *International Conference on Logic Programming*, pages 489–503, 1994.
- [31] T. Sato and Y. Kameya. PRISM: A language for symbolic-statistical modeling. *Proceedings of IJCAI 97*, pages 1330–1335, 1997.
- [32] T. Sato and Y. Kameya. Statistical abduction with tabulation. *Kowalski 60th birthday volume*, 2001.