2

Probabilistic Logic Programming Languages

Various approaches have been proposed for combining logic programming with probability theory. They can be broadly classified into two categories: those based on the *Distribution Semantics* (DS) [Sato, 1995] and those that follow a *Knowledge Base Model Construction* (KBMC) approach.

For languages in the first category, a probabilistic logic program without function symbols defines a probability distribution over normal logic programs (termed *worlds*). To define the probability of a query, this distribution is extended to a joint distribution of the query and the worlds and the probability of the query is obtained from the joint distribution by marginalization, i.e., by summing out the worlds. For probabilistic logic programs with function symbols, the definition is more complex, see Chapter 3.

The distribution over programs is defined by encoding random choices for clauses. Each choice generates an alternative version of the clause and the set of choices is associated with a probability distribution. The various languages that follow the DS differ in how the choices are encoded. In all languages, however, choices are independent from each other.

In the KBMC approach, instead, a probabilistic logic program is a compact way of encoding a large graphical model, either a BN or MN. In the KBMC approach, the semantics of a program is defined by the method for building the graphical model from the program.

2.1 Languages with the Distribution Semantics

The languages following DS differ in how they encode choices for clauses, and how the probabilities for these choices are stated. As will be shown in Section 2.4, they all have the same expressive power. This fact shows that the differences in the languages are syntactic, and also justifies speaking of *the* DS.

2.1.1 Logic Programs with Annotated Disjunctions

In Logic Programs with Annotated Disjunctions (LPADs) [Vennekens et al., 2004], the alternatives are expressed by means of annotated disjunctive heads of clauses. An *annotated disjunctive clause* C_i has the form

$$h_{i1} : \Pi_{i1} \ ; \ \ldots \ ; h_{in_i} : \Pi_{in_i} \leftarrow b_{i1}, \ldots, b_{im_i}$$

where $h_{i1}, \ldots, h_{in_i}$ are logical atoms, $b_{i1}, \ldots, b_{im_i}$ are logical literals, and $\Pi_{i1}, \ldots, \Pi_{in_i}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} = 1$. An LPAD is a finite set of annotated disjunctive clauses.

Each world is obtained by selecting one atom from the head of each grounding of each annotated disjunctive clause.

Example 12 (Medical symptoms – LPAD). *The following LPAD models the appearance of medical symptoms as a consequence of disease. A person may sneeze if he has the flu or if he has hay fever:*

$$sneezing(X) : 0.7 \ ; \ null : 0.3 \leftarrow flu(X).$$
$$sneezing(X) : 0.8 \ ; \ null : 0.2 \leftarrow hay\_fever(X).$$
$$flu(bob).$$
$$hay\_fever(bob).$$

The first clause can be read as: if X has the flu, then X sneezes with probability 0.7 and nothing happens with probability 0.3. Similarly, the second clause can be read as: if X has hay fever, then X sneezes with probability 0.8 and nothing happens with probability 0.2. Here, and for the other languages based on the distribution semantics, the atom $null$ does not appear in the body of any clause and is used to represent an alternative in which no atom is selected. It can also be omitted obtaining

$$sneezing(X) : 0.7 \leftarrow flu(X).$$
$$sneezing(X) : 0.8 \leftarrow hay\_fever(X).$$
$$flu(bob).$$
$$hay\_fever(bob).$$

As can be seen from the example, LPADs encode in a natural way programs representing causal mechanisms: flu and hay fever are causes for sneezing, which, however, is probabilistic, in the sense that it may or may not happen even when the causes are present. The relationship between the DS, and LPADs in particular, and causal reasoning is discussed in Section 2.8.

2.1.2 ProbLog

The design of ProbLog [De Raedt et al., 2007] was motivated by the desire to make the simplest probabilistic extension of Prolog. In ProbLog, alternatives are expressed by *probabilistic facts* of the form

$$\Pi_i :: f_i$$

where $\Pi_i \in [0, 1]$ and f_i is an atom, meaning that each ground instantiation $f_i\theta$ of f_i is true with probability Π_i and false with probability $1 - \Pi_i$. Each world is obtained by selecting or rejecting each grounding of all probabilistic facts.

Example 13 (Medical symptoms – ProbLog). *Example 12 can be expressed in ProbLog as:*

$$\begin{array}{l} sneezing(X) \leftarrow flu(X), flu\_sneezing(X). \\ sneezing(X) \leftarrow hay\_fever(X), hay\_fever\_sneezing(X). \\ flu(bob). \\ hay\_fever(bob). \\ 0.7 :: flu\_sneezing(X). \\ 0.8 :: hay\_fever\_sneezing(X). \end{array}$$

2.1.3 Probabilistic Horn Abduction

Probabilistic Horn Abduction (PHA) [Poole, 1993b] and Independent Choice Logic (ICL) [Poole, 1997] express alternatives by facts, called *disjoint statements*, having the form

$$disjoint([a_{i1} : \Pi_{i1}, \ldots, a_{in} : \Pi_{in_i}]).$$

where each a_{ik} is a logical atom and each Π_{ik} a number in $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} = 1$. Such a statement can be interpreted in terms of its ground instantiations: for each substitution θ grounding the atoms of the statement, the $a_{ik}\theta$s are random alternatives and $a_{ik}\theta$ is true with probability Π_{ik}. Each world is obtained by selecting one atom from each grounding of each disjoint statement in the program. In practice, each ground instantiation of a disjoint statement corresponds to a random variable with as many values as the alternatives in the statement.

Example 14 (Medical symptoms – ICL). *Example 12 can be expressed in ICL as:*

$$sneezing(X) \leftarrow flu(X), flu\_sneezing(X).$$
$$sneezing(X) \leftarrow hay\_fever(X), hay\_fever\_sneezing(X).$$
$$flu(bob).$$
$$hay\_fever(bob).$$

$$disjoint([flu\_sneezing(X) : 0.7, null : 0.3]).$$
$$disjoint([hay\_fever\_sneezing(X) : 0.8, null : 0.2]).$$

In ICL, LPADs, and ProbLog, each grounding of a probabilistic clause is associated with a random variable with as many values as alternatives/head disjuncts for ICL and LPADs and with two values for ProbLog. The random variables corresponding to different instantiations of a probabilistic clause are independent and identically distributed (IID).

2.1.4 PRISM

The language PRISM [Sato and Kameya, 1997] is similar to PHA/ICL but introduces random facts via the predicate $msw/3$ (*multi-switch*):

$$msw(SwitchName, TrialId, Value).$$

The first argument of this predicate is a *random switch name*, a term representing a set of discrete random variables; the second argument is an integer, the *trial id*; and the third argument represents a value for that variable. The set of possible values for a switch is defined by a fact of the form

$$values(SwitchName, [v_1, \ldots, v_n]).$$

where $SwitchName$ is again a term representing a switch name and each v_i is a term. Each ground pair $(SwitchName, TrialId)$ represents a distinct random variable and the set of random variables associated with the same switch are IID.

The probability distribution over the values of the random variables associated with $SwitchName$ is defined by a directive of the form

$$\leftarrow set\_sw(SwitchName, [\Pi_1, \ldots, \Pi_n]).$$

where p_i is the probability that variable $SwitchName$ takes value v_i. Each world is obtained by selecting one value for each trial id of each random switch.

Example 15 (Coin tosses – PRISM). *The modeling of coin tosses shows differences in how the various PLP languages represent IID random variables. Suppose that coin c_1 is known not to be fair, but that all tosses of c_1 have the same probabilities of outcomes – in other words, each toss of c_1 is taken from a family of IID random variables. This can be represented in PRISM as*

$$values(c_1, [head, tail]).$$
$$\leftarrow set\_sw(c_1, [0.4, 0.6])$$

Different tosses of c_1 can then be identified using the trial id *argument of $msw/3$.*

In PHA/ICL and many other PLP languages, each ground instantiation of a disjoint/1 *statement represents a distinct random variable, so that IID random variables need to be represented through the statement's instantiation patterns: e.g.,*

$$disjoint([coin(c_1, TossNumber, head) : 0.4,$$
$$coin(c_1, TossNumber, tail) : 0.6]).$$

In practice, the PRISM system accepts an $msw/2$ predicate whose atoms do not contain the trial id and for which each occurrence in a program is considered as being associated with a different new variable.

Example 16 (Medical symptoms – PRISM). *Example 14 can be encoded in PRISM as:*

$sneezing(X) \leftarrow flu(X), msw(flu\_sneezing(X), 1).$
$sneezing(X) \leftarrow hay\_fever(X), msw(hay\_fever\_sneezing(X), 1).$
$flu(bob).$
$hay\_fever(bob).$

$values(flu\_sneezing(\_X), [1, 0]).$
$values(hay\_fever\_sneezing(\_X), [1, 0]).$
$\leftarrow set\_sw(flu\_sneezing(\_X), [0.7, 0.3]).$
$\leftarrow set\_sw(hay\_fever\_sneezing(\_X), [0.8, 0.2]).$

2.2 The Distribution Semantics for Programs Without Function Symbols

We present first the DS for the case of ProbLog as it is the language with the simplest syntax. A ProbLog program $\mathcal{P}$ is composed by a set of normal

rules $\mathcal{R}$ and a set $\mathcal{F}$ of probabilistic facts. Each *probabilistic fact* is of the form $\Pi_i :: f_i$ where $\Pi_i \in [0, 1]$ and f_i is an atom[1], meaning that each ground instantiation $f_i\theta$ of f_i is true with probability Π_i and false with probability $1 - \Pi_i$. Each world is obtained by selecting or rejecting each grounding of each probabilistic fact.

An *atomic choice* indicates whether grounding $f\theta$ of a probabilistic fact $F = p :: f$ is selected or not. It is represented with the triple (f, θ, k) where $k \in \{0, 1\}$ and $k = 1$ means that the fact is selected, $k = 0$ that it is not. A set κ of atomic choices is *consistent* if it does not contain two atomic choices (f, θ, k) and (f, θ, j) with $k \neq j$ (only one alternative is selected for a ground probabilistic fact). The function $consistent(\kappa)$ returns true if κ is consistent. A *composite choice* κ is a consistent set of atomic choices. The probability of composite choice κ is

$$P(\kappa) = \prod_{(f_i,\theta,1)\in\kappa} \Pi_i \prod_{(f_i,\theta,0)\in\kappa} 1 - \Pi_i.$$

A *selection* σ is a total composite choice, i.e., contains one atomic choice for every grounding of every probabilistic fact. A *world* w_σ is a logic program that is identified by a selection σ. The world w_σ is formed by including the atom corresponding to each atomic choice $(f, \theta, 1)$ of σ.

The probability of a world w_σ is $P(w_\sigma) = P(\sigma)$. Since in this section we are assuming programs without function symbols, the set of groundings of each probabilistic fact is finite, and so is the set of worlds $W_\mathcal{P}$. Accordingly, for a ProbLog program $\mathcal{P}$, $W_\mathcal{P} = \{w_1, \ldots, w_m\}$. Moreover, $P(w)$ is a distribution over worlds: $\sum_{w\in W_\mathcal{P}} P(w) = 1$. We call *sound* a program for which every world has a two-valued WFM. We consider here sound programs, for non-sound ones, see Section 2.9.

Let q be a query in the form of a ground atom. We define the conditional probability of q given a world w as: $P(q|w) = 1$ if q is true in w and 0 otherwise. Since the program is sound, q can be only true or false in a world. The probability of q can thus be computed by summing out the worlds from the joint distribution of the query and the worlds:

$$P(q) = \sum_w P(q, w) = \sum_w P(q|w)P(w) = \sum_{w\models q} P(w). \tag{2.1}$$

This formula can also be used for computing the probability of a conjunction $q_1, \ldots, q_n$ of ground atoms since the truth of a conjunction of ground atoms

[1]With an abuse of notation, sometimes we use $\mathcal{F}$ to indicate the set containing the atoms f_is. The meaning of $\mathcal{F}$ will be clear from the context.

in a world is well defined. So we can compute the conditional probability of a query q given evidence e in the form of a conjunction of ground atoms $e_1, \ldots, e_m$ as

$$P(q|e) = \frac{P(q,e)}{P(e)} \tag{2.2}$$

We can also assign a probability to a query q by defining a probability space. Since $W_\mathcal{P}$ is finite, then $(W_\mathcal{P}, \mathbb{P}(W_\mathcal{P}))$ is a measurable space. For an element $\omega \in \mathbb{P}(W_\mathcal{P})$, define $\mu(\omega)$ as

$$\mu(\omega) = \sum_{w \in \omega} P(w)$$

with the probability of a world $P(w)$ defined as above. Then it's easy to see that $(W_\mathcal{P}, \mathbb{P}(W_\mathcal{P}), \mu)$ is a finitely additive probability space.

Given a ground atom q, define the function $Q : W_\mathcal{P} \to \{0,1\}$ as

$$Q(w) = \begin{cases} 1 & \text{if } w \models q \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Since the set of events is the powerset, then $Q^{-1}(\gamma) \in \mathbb{P}(W_\mathcal{P})$ for all $\gamma \subseteq \{0,1\}$ and Q is a random variable. The distribution of Q is defined by $P(Q=1)$ ($P(Q=0)$ is given by $1 - P(Q=1)$) and we indicate $P(Q=1)$ with $P(q)$.

We can now compute $P(q)$ as

$$P(q) = \mu(Q^{-1}(\{1\})) = \mu(\{w | w \in W_\mathcal{P}, w \models q\}) = \sum_{w \models q} P(w)$$

obtaining the same formula as Equation (2.1).

The distribution over worlds also induces a distribution over interpretations: given an interpretation I, we can define the conditional probability of I given a world w as: $P(I|w) = 1$ is I is the model of w ($I \models w$) and 0 otherwise. The distribution over interpretations is then given by a formula similar to Equation (2.1):

$$P(I) = \sum_w P(I,w) = \sum_w P(I|w)P(w) = \sum_{I \models w} P(w) \tag{2.4}$$

We call the interpretations I for which $P(I) > 0$ *possible models* because they are models for at least one world.

Now define the function $\mathbf{I} : W_{\mathcal{P}} \rightarrow \{0, 1\}$ as

$$\mathbf{I}(I) = \begin{cases} 1 & \text{if } I \models w \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

$\mathbf{I}^{-1}(\gamma) \in \mathbb{P}(W_{\mathcal{P}})$ for all $\gamma \subseteq \{0, 1\}$ so $\mathbf{I}$ is a random variable for probability space $(W_{\mathcal{P}}, \mathbb{P}(W_{\mathcal{P}}), \mu)$. The distribution of $\mathbf{I}$ is defined by $P(\mathbf{I} = 1)$ and we indicate $P(\mathbf{I} = 1)$ with $P(I)$.

We can now compute $P(I)$ as

$$P(I) = \mu(\mathbf{I}^{-1}(\{1\})) = \mu(\{w | w \in W_{\mathcal{P}}, I \models w\}) = \sum_{I \models w} P(w)$$

obtaining the same formula as Equation (2.4).

The probability of a query q can be obtained from the distribution over interpretations by defining the conditional probability of q given an interpretation I as $P(q|I) = 1$ if $I \models q$ and 0 otherwise and by marginalizing the interpretations obtaining

$$P(q) = \sum_{I} P(q, I) = \sum_{I} P(q|I)P(I) = \sum_{I \models q} P(I) = \sum_{I \models q, I \models w} P(w) \tag{2.6}$$

So the probability of a query can be obtained by summing the probability of the possible models where the query is true.

Example 17 (Medical symptoms – worlds – ProbLog). *Consider the program of Example 13. The program has four worlds*

$$\begin{array}{rl} w_1 = \{ & \\ & \mathit{flu\_sneezing}(\mathit{bob}). \\ & \mathit{hay\_fever\_sneezing}(\mathit{bob}). \\ \} & \\ P(w_1) = & 0.7 \times 0.8 \end{array} \qquad \begin{array}{rl} w_2 = \{ & \\ & \mathit{hay\_fever\_sneezing}(\mathit{bob}). \\ \} & \\ P(w_2) = & 0.3 \times 0.8 \end{array}$$

$$\begin{array}{rl} w_3 = \{ & \\ & \mathit{flu\_sneezing}(\mathit{bob}). \\ \} & \\ P(w_3) = & 0.7 \times 0.2 \end{array} \qquad \begin{array}{rl} w_4 = \{ & \\ \} & \\ P(w_4) = & 0.3 \times 0.2 \end{array}$$

The query $\mathit{sneezing}(\mathit{bob})$ *is true in three worlds and its probability*

$$P(\mathit{sneezing}(\mathit{bob})) = 0.7 \times 0.8 + 0.3 \times 0.8 + 0.7 \times 0.2 = 0.94.$$

Note that the contributions from the two clauses are combined disjunctively. The probability of the query is thus computed using the rule giving the probability of the disjunction of two independent Boolean random variables:

$$P(a \vee b) = P(a) + P(b) - P(a)P(b) = 1 - (1 - P(a))(1 - P(b)).$$

In our case, $P(sneezing(bob)) = 0.7 + 0.8 - 0.7 \cdot 0.8 = 0.94$.

We now give the semantics for LPADs. A clause

$$C_i = h_{i1} : \Pi_{i1} \ ; \ \ldots \ ; h_{in_i} : \Pi_{in_i} \leftarrow b_{i1}, \ldots, b_{im_i}$$

stands for a set of probabilistic clauses, one for each ground instantiation $C_i\theta$ of C_i. Each ground probabilistic clause represents a choice among n_i normal clauses, each of the form

$$h_{ik} \leftarrow b_{i1}, \ldots, b_{im_i}$$

for $k = 1 \ldots, n_i$. Moreover, another clause

$$null \leftarrow b_{i1}, \ldots, b_{im_i}$$

is implicitly encoded which is associated with probability $\Pi_0 = 1 - \sum_{k=1}^{n_i} \Pi_k$. So for LPAD P an *atomic choice* is the selection of a head atom for a grounding $C_i\theta_j$ of a probabilistic clause C_i, including the atom $null$. An atomic choice is represented in this case by the triple (C_i, θ_j, k), where θ_j is a grounding substitution and $k \in \{0, 1, \ldots, n_i\}$. An atomic choice represents an equation of the form $\mathrm{X}_{ij} = k$ where X_{ij} is a random variable associated with $C_i\theta_j$. The definition of consistent set of atomic choices, of composite choices, and of the probability of a composite choice is the same as for ProbLog. Again, a *selection* σ is a total composite choice (one atomic choice for every grounding of each probabilistic clause). A selection σ identifies a logic program w_σ (a *world*) that contains the normal clauses obtained by selecting head atom $h_{ik}\theta$ for each atomic choice (C_i, θ, k):

$$\begin{aligned} w_\sigma = \{ \quad & (h_{ik} \leftarrow b_{i1}, \ldots, b_{im_i})\theta | (C_i, \theta, k) \in \sigma, \\ & C_i = h_{i1} : \Pi_{i1} \ ; \ \ldots \ ; h_{in_i} : \Pi_{in_i} \leftarrow b_{i1}, \ldots, b_{im_i}, C_i \in \mathcal{P}\} \end{aligned}$$

As for ProbLog, the probability of w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$, the set of worlds $W_P = \{w_1, \ldots, w_m\}$ is finite, and $P(w)$ is a distribution over worlds.

If q is a query, we can define $P(q|w)$ as for ProbLog and again the probability of q is given by Equation (2.1)

Example 18 (Medical symptoms – worlds – LPAD). *The LPAD of Example 12 has four worlds:*

$$
\begin{aligned}
w_1 = \{ & \\
& sneezing(bob) \leftarrow \mathit{flu}(bob). \\
& sneezing(bob) \leftarrow hay\_fever(bob). \\
& \mathit{flu}(bob). \quad hay\_fever(bob). \\
\} & \\
P(w_1) = \; & 0.7 \times 0.8
\end{aligned}
$$

$$
\begin{aligned}
w_2 = \{ & \\
& null \leftarrow \mathit{flu}(bob). \\
& sneezing(bob) \leftarrow hay\_fever(bob). \\
& \mathit{flu}(bob). \quad hay\_fever(bob). \\
\} & \\
P(w_2) = \; & 0.3 \times 0.8
\end{aligned}
$$

$$
\begin{aligned}
w_3 = \{ & \\
& sneezing(bob) \leftarrow \mathit{flu}(bob). \\
& null \leftarrow hay\_fever(bob). \\
& \mathit{flu}(bob). \quad hay\_fever(bob). \\
\} & \\
P(w_3) = \; & 0.7 \times 0.2
\end{aligned}
$$

$$
\begin{aligned}
w_4 = \{ & \\
& null \leftarrow \mathit{flu}(bob). \\
& null \leftarrow hay\_fever(bob). \\
& \mathit{flu}(bob). \quad hay\_fever(bob). \\
\} & \\
P(w_4) = \; & 0.3 \times 0.2
\end{aligned}
$$

$sneezing(bob)$ *is true in three worlds and its probability is*

$$P(sneezing(bob)) = 0.7 \times 0.8 + 0.3 \times 0.8 + 0.7 \times 0.2 = 0.94$$

2.3 Examples of Programs

In this section, we provide some examples of programs to better illustrate the syntax and the semantics.

Example 19 (Detailed medical symptoms – LPAD). *The following LPAD[2] models a program that describe medical symptoms in a way that is slightly more elaborated than Example 12:*

$$strong\_sneezing(X) : 0.3 \; ; \; moderate\_sneezing(X) : 0.5 \leftarrow flu(X).$$
$$strong\_sneezing(X) : 0.2 \; ; \; moderate\_sneezing(X) : 0.6 \leftarrow hay\_fever(X).$$
$$flu(bob).$$
$$hay\_fever(bob).$$

Here the clauses have three alternatives in the head of which the one associated with atom $null$ is left implicit. This program has nine worlds, the query $strong\_sneezing(bob)$ is true in five of them, and $P(strong\_sneezing(bob)) = 0.44$.

Example 20 (Coin – LPAD). *The coin example of [Vennekens et al., 2004] is represented as[3]:*

$$heads(Coin) : 1/2 \; ; \; tails(Coin) : 1/2 \leftarrow toss(Coin), \sim biased(Coin).$$
$$heads(Coin) : 0.6 \; ; \; tails(Coin) : 0.4 \leftarrow toss(Coin), biased(Coin).$$
$$fair(Coin) : 0.9 \; ; \; biased(Coin) : 0.1.$$
$$toss(coin).$$

The first clause states that, if we toss a coin that is not biased, it has equal probability of landing heads and tails. The second states that, if the coin is biased, it has a slightly higher probability of landing heads. The third states that the coin is fair with probability 0.9 and biased with probability 0.1 and the last clause states that we toss the coin with certainty. This program has eight worlds, the query $heads(coin)$ is true in four of them, and its probability is 0.51.

Example 21 (Eruption – LPAD). *Consider this LPAD[4] from Riguzzi and Di Mauro [2012] that is inspired by the morphological characteristics of the Italian island of Stromboli:*

[2] http://cplint.eu/e/sneezing.pl
[3] http://cplint.eu/e/coin.pl
[4] http://cplint.eu/e/eruption.pl

$C_1 =$ $eruption : 0.6$ *;* $earthquake : 0.3$ *:-* $sudden\_energy\_release$, $fault\_rupture(X)$.
$C_2 =$ $sudden\_energy\_release : 0.7$.
$C_3 =$ $fault\_rupture(southwest\_northeast)$.
$C_4 =$ $fault\_rupture(east\_west)$.

The island of Stromboli is located at the intersection of two geological faults, one in the southwest–northeast direction, the other in the east–west direction, and contains one of the three volcanoes that are active in Italy. This program models the possibility that an eruption or an earthquake occurs at Stromboli. If there is a sudden energy release under the island and there is a fault rupture, then there can be an eruption of the volcano on the island with probability 0.6 or an earthquake in the area with probability 0.3. The energy release occurs with probability 0.7 and we are sure that ruptures occur in both faults.

Clause C_1 *has two groundings,* $C_1\theta_1$ *with*

$$\theta_1 = \{X/southwest\_northeast\}$$

and $C_1\theta_2$ *with*

$$\theta_2 = \{X/east\_west\},$$

while clause C_2 *has a single grounding* $C_2\varnothing$*. Since* C_1 *has three head atoms and* C_2 *two, the program has* $3 \times 3 \times 2$ *worlds. The query* $eruption$ *is true in five of them and its probability is* $P(eruption) = 0.6 \cdot 0.6 \cdot 0.7 + 0.6 \cdot 0.3 \cdot 0.7 + 0.6 \cdot 0.1 \cdot 0.7 + 0.3 \cdot 0.6 \cdot 0.7 + 0.1 \cdot 0.6 \cdot 0.7 = 0.588$.

Example 22 (Monty Hall puzzle – LPAD). *The Monty Hall puzzle [Baral et al., 2009] refers to the TV game show hosted by Monty Hall in which a player has to choose which of three closed doors to open. Behind one door, there is a prize, while behind the other two, there is nothing. Once the player has selected the door, Monty Hall opens one of the remaining closed doors which does not contain the prize, and then he asks the player if he would like to change his door with the other closed door or not. The problem of this game is to determine whether the player should switch. The following program provides a solution*[5]*. The prize is behind one of the three doors with the same probability:*

$prize(1) : 1/3$ *;* $prize(2) : 1/3$ *;* $prize(3) : 1/3$.

The player has selected door 1:

$selected(1)$.

[5] http://cplint.eu/e/monty.swinb

Monty opens door 2 with probability 0.5 and door 3 with probability 0.5 if the prize is behind door 1:

$open\_door(2) : 0.5$ *;* $open\_door(3) : 0.5 \leftarrow prize(1).$

Monty opens door 2 if the prize is behind door 3:

$open\_door(2) \leftarrow prize(3).$

Monty opens door 3 if the prize is behind door 2:

$open\_door(3) \leftarrow prize(2).$

The player keeps his choice and wins if he has selected a door with the prize:

$win\_keep \leftarrow prize(1).$

The player switches and wins if the prize is behind the door that he has not selected and that Monty did not open:

$win\_switch \leftarrow prize(2), open\_door(3).$

$win\_switch \leftarrow prize(3), open\_door(2).$

Querying win_keep *and* win_switch *we obtain probability 1/3 and 2/3 respectively, so the player should switch. Note that if you change the probability distribution of Monty selecting a door to open when the prize is behind the door selected by the player, then the probability of winning by switching remains the same.*

Example 23 (Three-prisoner puzzle – LPAD). *The following program*[6] *from [Riguzzi et al., 2016a] encodes the three-prisoner puzzle. In Grünwald and Halpern [2003], the problem is described as:*

> *Of three prisoners* a*,* b*, and* c*, two are to be executed, but* a *does not know which. Thus,* a *thinks that the probability that* i *will be executed is 2/3 for* $i \in \{a, b, c\}$*. He says to the jailer, "Since either* b *or* c *is certainly going to be executed, you will give me no information about my own chances if you give me the name of one man, either* b *or* c*, who is going to be executed." But then, no matter what the jailer says, naive conditioning leads* a *to believe that his chance of execution went down from 2/3 to 1/2.*

Each prisoner is safe with probability 1/3:

$safe(a) : 1/3$ *;* $safe(b) : 1/3$ *;* $safe(c) : 1/3.$

If a *is safe, the jailer tells that one of the other prisoners will be executed uniformly at random:*

$tell\_executed(b) : 1/2$ *;* $tell\_executed(c) : 1/2 \leftarrow safe(a).$

Otherwise, he tells that the only unsafe prisoner will be executed:

[6] http://cplint.eu/e/jail.swinb

$tell\_executed(b) \leftarrow safe(c).$
$tell\_executed(c) \leftarrow safe(b).$

The jailer speaks if he tells that somebody will be executed:

$tell \leftarrow tell\_executed(\_).$

a is safe after the jailer utterance if he is safe and the jailer speaks:

$safe\_after\_tell : -safe(a), tell.$

By computing the probability of $safe(a)$ and $safe\_after\_tell$, we get the same probability of 1/3, so the jailer utterance does not change the probability of a of being safe.

We can see this also by considering conditional probabilities: the probability of $safe(a)$ given the jailer utterance $tell$ is

$$P(safe(a)|tell) = \frac{P(safe(a), tell)}{P(tell)} = \frac{P(safe\_after\_tell)}{P(tell)} = \frac{1/3}{1} = 1/3$$

because the probability of $tell$ is 1.

Example 24 (Russian roulette with two guns – LPAD). *The following example*[7] *models a Russian roulette game with two guns [Baral et al., 2009]. The death of the player is caused with probability 1/6 by triggering the left gun and similarly for the right gun:*

$death : 1/6 \leftarrow pull\_trigger(left\_gun).$
$death : 1/6 \leftarrow pull\_trigger(right\_gun).$
$pull\_trigger(left\_gun).$
$pull\_trigger(right\_gun).$

Querying the probability of $death$ we gent the probability of the player of dying.

Example 25 (Mendelian rules of inheritance – LPAD). *Blockeel [2004] presents a program*[8] *that encodes the Mendelian rules of inheritance of the color of pea plants. The color of a pea plant is determined by a gene that exists in two forms (alleles), purple,* p*, and white,* w*. Each plant has two alleles for the color gene that reside on a couple of chromosomes.* cg(X,N,A) *indicates that plant* X *has allele* A *on chromosome* N*. The program is:*

$color(X, white) \leftarrow cg(X, 1, w), cg(X, 2, w).$
$color(X, purple) \leftarrow cg(X, \_A, p).$

[7] http://cplint.eu/e/trigger.pl
[8] http://cplint.eu/e/mendel.pl

$cg(X,1,A):0.5\ ;\ cg(X,1,B):0.5 \leftarrow$
$\quad mother(Y,X), cg(Y,1,A), cg(Y,2,B).$
$cg(X,2,A):0.5\ ;\ cg(X,2,B):0.5 \leftarrow$
$\quad father(Y,X), cg(Y,1,A), cg(Y,2,B).$
$mother(m,c).\quad father(f,c).$
$cg(m,1,w).\quad cg(m,2,w).\quad cg(f,1,p).\quad cg(f,2,w).$

The facts of the program express that c *is the offspring of* m *and* f *and that the alleles of* m *are* ww *and of* f *are* pw. *The disjunctive rules encode the fact that an offspring inherits the allele on chromosome 1 from the mother and the allele on chromosome 2 from the father. In particular, each allele of the parent has a probability of 50% of being transmitted. The definite clauses for* color *express the fact that the color of a plant is purple if at least one of the alleles is* p*, i.e., that the* p *allele is dominant. In a similar way, the rules of blood type inheritance can be written in an LPAD*[9].

Example 26 (Path probability – LPAD). *An interesting application of PLP under the DS is the computation of the probability of a path between two nodes in a graph in which the presence of each edge is probabilistic*[10]*:*

$path(X,X).$
$path(X,Y) \leftarrow path(X,Z), edge(Z,Y).$
$edge(a,b):0.3.\quad edge(b,c):0.2.\quad edge(a,c):0.6.$

This program, coded in ProbLog, was used in [De Raedt et al., 2007] for computing the probability that two biological concepts are related in the BIOMINE network [Sevon et al., 2006].

PLP under the DS can encode BNs Vennekens et al. [2004]: each value of each random variable is encoded by a ground atom, each row of each CPT is encoded by a rule with the value of parents in the body and the probability distribution of values of the child in the head.

Example 27 (Alarm BN – LPAD). *For example, the BN of Example 10 that we repeat in Figure 2.1 for readability can be encoded with the program*[11]

[9] http://cplint.eu/e/bloodtype.pl
[10] http://cplint.eu/e/path.swinb
[11] http://cplint.eu/e/alarm.pl

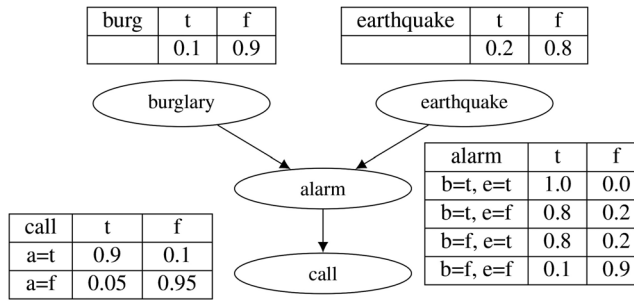


Figure 2.1 Example of a BN.

$burg(t) : 0.1$ *;* $burg(f) : 0.9.$
$earthquake(t) : 0.2$ *;* $earthquake(f) : 0.8.$
$alarm(t) \leftarrow burg(t), earthq(t).$
$alarm(t) : 0.8$ *;* $alarm(f) : 0.2 \leftarrow burg(t), earthq(f).$
$alarm(t) : 0.8$ *;* $alarm(f) : 0.2 \leftarrow burg(f), earthq(t).$
$alarm(t) : 0.1$ *;* $alarm(f) : 0.9 \leftarrow burg(f), earthq(f).$
$call(t) : 0.9$ *;* $call(f) : 0.1 \leftarrow alarm(t).$
$call(t) : 0.05$ *;* $call(f) : 0.95 \leftarrow alarm(f).$

2.4 Equivalence of Expressive Power

To show that all these languages have the same expressive power, we discuss transformations among probabilistic constructs from the various languages.

The mapping between PHA/ICL and PRISM translates each PHA/ICL disjoint statement to a multi-switch declaration and vice versa in the obvious way. The mapping from PHA/ICL and PRISM to LPADs translates each disjoint statement/multi-switch declaration to a disjunctive LPAD fact.

The translation from an LPAD into PHA/ICL (first shown in [Vennekens and Verbaeten, 2003]) rewrites each clause C_i with v variables $\overline{X}$

$$h_1 : \Pi_1 \; ; \; \ldots \; ; h_n : \Pi_n \leftarrow B.$$

into PHA/ICL by adding n new predicates $\{choice_{i1}/v, \ldots, choice_{in}/v\}$ and a disjoint statement:

$$h_1 \leftarrow B, choice_{i1}(\overline{X}).$$
$$\vdots$$
$$h_n \leftarrow B, choice_{in}(\overline{X}).$$

$$disjoint([choice_{i1}(\overline{X}) : \Pi_1, \ldots, choice_{in}(\overline{X}) : \Pi_n]).$$

For instance, the first clause of the medical symptoms LPAD of Example 19 is translated to

$$strong\_sneezing(X) \leftarrow flu(X), choice_{11}(X).$$
$$moderate\_sneezing(X) : 0.5 \leftarrow flu(X), choice_{12}(X).$$
$$disjoint([choice_{11}(X) : 0.3, choice_{12}(X) : 0.5, choice_{13} : 0.2]).$$

where the clause $null \leftarrow flu(X), choice_{13}.$ is omitted since null does not appear in the body of any clause.

Finally, as shown in [De Raedt et al., 2008], to convert LPADs into ProbLog, each clause C_i with v variables $\overline{X}$

$$h_1 : \Pi_1 \ ; \ \ldots \ ; h_n : \Pi_n \leftarrow B.$$

is translated into ProbLog by adding $n-1$ probabilistic facts for predicates $\{f_{i1}/v, \ldots, f_{in}/v\}$:

$$h_1 \leftarrow B, f_{i1}(\overline{X}).$$
$$h_2 \leftarrow B, \sim f_{i1}(\overline{X}), f_{i2}(\overline{X}).$$
$$\vdots$$
$$h_n \leftarrow B, \sim f_{i1}(\overline{X}), \ldots, \sim f_{in-1}(\overline{X}).$$

$$\pi_1 :: f_{i1}(\overline{X}).$$
$$\vdots$$
$$\pi_{n-1} :: f_{in-1}(\overline{X}).$$

where

$$\pi_1 = \Pi_1$$
$$\pi_2 = \frac{\Pi_2}{1-\pi_1}$$
$$\pi_3 = \frac{\Pi_3}{(1-\pi_1)(1-\pi_2)}$$
$$\ldots$$

In general

$$\pi_i = \frac{\Pi_i}{\prod_{j=1}^{i-1}(1-\pi_j)}.$$

Note that while the translation into ProbLog introduces negation, the introduced negation involves only probabilistic facts, and so the transformed program will have a two-valued model whenever the original program does.

For instance, the first clause of the medical symptoms LPAD of Example 19 is translated to

$$strong\_sneezing(X) \leftarrow flu(X), f_{11}(X).$$
$$moderate\_sneezing(X) : 0.5 \leftarrow flu(X), \sim f_{11}(X), f_{12}(X).$$
$$0.3 :: f_{11}(X).$$
$$0.71428571428 :: f_{12}(X).$$

2.5 Translation to Bayesian Networks

We discuss here how an acyclic ground LPAD can be translated to a BN. Let us first define the acyclic property for LPADs, extending Definition 4. An LPAD is *acyclic* if an integer level can be assigned to each ground atom so that the level of each atom in the head of each ground rule is the same and is higher than the level of each atom in the body.

An acyclic ground LPAD $\mathcal{P}$ can be translated to a BN $\beta(\mathcal{P})$ [Vennekens et al., 2004]. $\beta(\mathcal{P})$ is built by associating each atom a in $\mathcal{B}_\mathcal{P}$ with a binary variable a with values true (1) and false (0). Moreover, for each rule C_i of the following form

$h_1 : \Pi_1 \ ; \ \ldots \ ; h_n : \Pi_n \leftarrow b_1, \ldots b_m, \sim c_1 \ldots, \sim c_l$

in $ground(\mathcal{P})$, we add a new variable ch_i (for "choice for rule C_i") to $\beta(\mathcal{P})$. ch_i has $b_1, \ldots, b_m$, $c_1, \ldots, c_l$ as parents. The values for ch_i are h_1, ..., h_n and $null$, corresponding to the head atoms. The CPT of ch_i is

	...	$b_1 = 1, \ldots, b_m = 1, c_1 = 0, \ldots, c_l = 0$	...
$\mathrm{ch}_i = h_1$	0.0	Π_1	0.0
...			
$\mathrm{ch}_n = h_n$	0.0	Π_n	0.0
$\mathrm{ch}_i = null$	1.0	$1 - \sum_{i=1}^{n} \Pi_i$	1.0

that can be expressed as

$$P(\mathrm{ch}_i | b_1, \ldots, c_l) \ = \ \begin{cases} \Pi_k & \text{if } \mathrm{ch}_i = h_k, b_i = 1, \ldots, c_l = 0 \\ 1 - \sum_{j=1}^{n} \Pi_j & \text{if } \mathrm{ch}_i = null, b_i = 1, \ldots, c_l = 0 \\ 1 & \text{if } \mathrm{ch}_i = null, \neg(b_i = 1, \ldots, c_l = 0) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

If the body is empty, the CPT for ch_i is

$\mathrm{ch}_i = h_1$	Π_1
$\ldots$	
$\mathrm{ch}_n = h_n$	Π_n
$\mathrm{ch}_i = null$	$1 - \sum_{i=1}^{n} \Pi_i$

Moreover, for each variable a corresponding to atom $a \in \mathcal{B}_P$, the parents are all the variables ch_i of rules C_i that have a in the head. The CPT for a is the following deterministic table:

	At least one parent equal to a	Remaining columns
$a = 1$	1.0	0.0
$a = 0$	0.0	1.0

encoding the function

$$a = f(\mathbf{ch}_a) = \begin{cases} 1 & \text{if } \exists \mathrm{ch}_i \in \mathbf{ch}_a : \mathrm{ch}_i = a \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{ch}_a$ are the parents of a. Note that in order to convert an LPAD containing variables into a BN, its grounding must be generated.

Example 28 (LPAD to BN). *Consider the following LPAD $\mathcal{P}$:*

$C_1 = a_1 : 0.4 \; ; \; a_2 : 0.3.$
$C_2 = a_2 : 0.1 \; ; \; a_3 : 0.2.$
$C_3 = a_4 : 0.6 \; ; \; a_5 : 0.4 \leftarrow a_1.$
$C_4 = a_5 : 0.4 \leftarrow a_2, a_3.$
$C_5 = a_6 : 0.3 \; ; \; a_7 : 0.2 \leftarrow a_2, a_5.$

Its corresponding network $\beta(\mathcal{P})$ is shown in Figure 1.7, where the CPT for a_2 and ch_5 are shown in Tables 2.1 and 2.2 respectively.

Table 2.1 Conditional probability table for a_2

$\mathrm{ch}_1, \mathrm{ch}_2$	a_1, a_2	a_1, a_3	a_2, a_2	a_2, a_3
$a_2 = 1$	1.0	0.0	1.0	1.0
$a_2 = 0$	0.0	1.0	0.0	0.0

Table 2.2 Conditional probability table for ch_5

a_2, a_5	1,1	1,0	0,1	0,0
$\mathrm{ch}_5 = x6$	0.3	0.0	0.0	0.0
$\mathrm{ch}_5 = x7$	0.2	0.0	0.0	0.0
$\mathrm{ch}_5 = null$	0.5	1.0	1.0	1.0

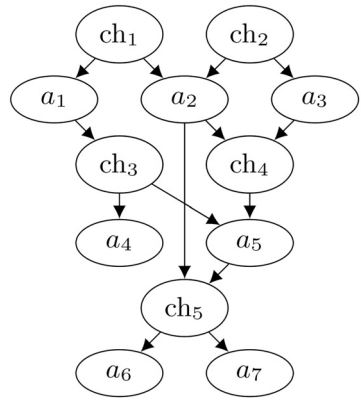


Figure 2.2 BN $\beta(\mathcal{P})$ equivalent to the program of Example 28.

An alternative translation $\gamma(\mathcal{P})$ for a ground program P is built by including random variables a for each atom a in $\mathcal{B}_\mathcal{P}$ and ch_i for each clause C_i as for $\beta(\mathcal{P})$. Moreover, $\gamma(\mathcal{P})$ includes the Boolean random variable body_i and the random variable X_i with values $h_1, \ldots, h_n$ and $null$ for each clause C_i.

The parents of body_i are $b_1, \ldots, b_m$, and $c_1, \ldots, c_l$ and its CPT encodes the deterministic AND Boolean function:

	$\ldots$	$b_1 = 1, \ldots, b_m = 1, c_1 = 0, \ldots, c_l = 0$	$\ldots$
$\mathrm{body}_i = 0$	1.0	0.0	1.0
$\mathrm{body}_i = 1$	0.0	1.0	0.0

If the body is empty, the CPT makes body_i surely true

$\mathrm{body}_i = 0$	0.0
$\mathrm{body}_i = 1$	1.0

X_i has no parents and has the CPT

$\mathrm{ch}_i = h_1$	Π_1
$\ldots$	
$\mathrm{ch}_i = h_n$	Π_n
$\mathrm{ch}_i = null$	$1 - \sum_{i=1}^{n} \Pi_i$

ch_i has X_i and body_i as parents with the deterministic CPT

$\mathrm{body}_i, \mathrm{X}_i$	$0, h_1$	$\ldots$	$0, h_n$	$0, null$	$1, h_1$	$\ldots$	$1, h_n$	$1, null$
$\mathrm{ch}_i = h_1$	0.0	$\ldots$	0.0	0.0	1.0	$\ldots$	0.0	0.0
$\ldots$								
$\mathrm{ch}_i = h_n$	0.0	$\ldots$	0.0	0.0	0.0	$\ldots$	1.0	0.0
$\mathrm{ch}_i = null$	1.0	$\ldots$	1.0	1.0	0.0	$\ldots$	0.0	1.0

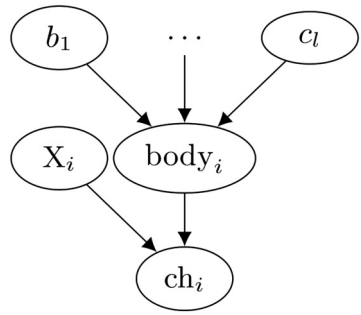


Figure 2.3 Portion of $\gamma(\mathcal{P})$ relative to a clause C_I.

encoding the function

$$\mathrm{ch}_i = f(\mathrm{body}_i, \mathrm{X}_i) = \begin{cases} \mathrm{X}_i & \text{if } \mathrm{body}_i = 1 \\ null & \text{if } \mathrm{body}_i = 0 \end{cases}$$

The parents of each variable a in $\gamma(\mathcal{P})$ are the variables ch_i of rules C_i that have a in the head as for $\beta(\mathcal{P})$, with the same CPT as in $\beta(\mathcal{P})$.

The portion of $\gamma(\mathcal{P})$ relative to a clause C_i is shown in Figure 2.3.

If we compute $P(\mathrm{ch}_i|b_1, \ldots, b_m, c_1, \ldots, c_l)$ by marginalizing

$$P(\mathrm{ch}_i, \mathrm{body}_i, \mathrm{X}_i|b_1, \ldots, b_m, c_1, \ldots, c_l)$$

we can see that we obtain the same dependency as in $\beta(\mathcal{P})$:

$$\begin{aligned}
&P(\mathrm{ch}_i|b_1, \ldots, c_l) = \\
&= \sum_{\mathrm{x}_i} \sum_{\mathrm{body}_i} P(\mathrm{ch}_i, \mathrm{body}_i, \mathrm{x}_i|b_1, \ldots, c_l) \\
&= \sum_{\mathrm{x}_i} \sum_{\mathrm{body}_i} P(\mathrm{ch}_i|\mathrm{body}_i, \mathrm{x}_i) P(\mathrm{x}_i) P(\mathrm{body}_i|b_1, \ldots, c_l) \\
&= \sum_{\mathrm{x}_i} P(\mathrm{x}_i) \sum_{\mathrm{body}_i} P(\mathrm{ch}_i|\mathrm{body}_i, \mathrm{x}_i) P(\mathrm{body}_i|b_1, \ldots, c_l) \\
&= \sum_{\mathrm{x}_i} P(\mathrm{x}_i) \sum_{\mathrm{body}_i} P(\mathrm{ch}_i|\mathrm{body}_i, \mathrm{x}_i) \begin{cases} 1 & \text{if } \mathrm{body}_i = 1, b_1 = 1, \ldots, c_l = 0 \\ 1 & \text{if } \mathrm{body}_i = 0, \neg(b_1 = 1, \ldots, c_l = 0) \\ 0 & \text{otherwise} \end{cases} \\
&= \sum_{\mathrm{x}_i} P(\mathrm{x}_i) \sum_{\mathrm{body}_i} \begin{cases} 1 & \text{if } \mathrm{ch}_i = \mathrm{x}_i, \mathrm{body}_i = 1, b_1 = 1, \ldots, c_l = 0 \\ 1 & \text{if } \mathrm{ch}_i = null, \mathrm{body}_i = 0, \neg(b_1 = 1, \ldots, c_l = 0) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

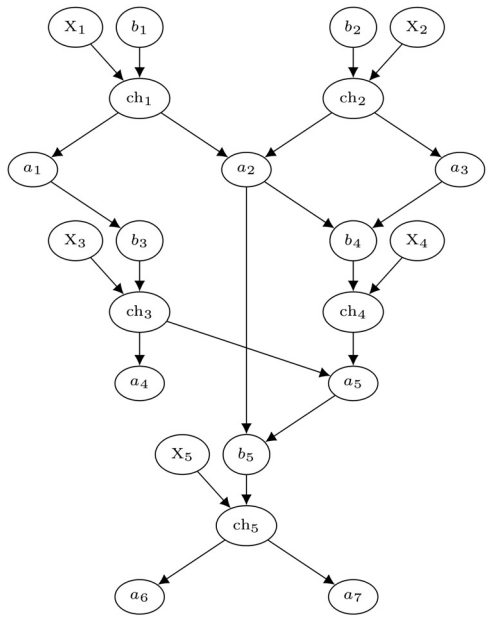


Figure 2.4 BN $\gamma(\mathcal{P})$ equivalent to the program of Example 28.

$$= \sum_{\mathrm{x}_i} P(\mathrm{x}_i) \begin{cases} 1 & \text{if } \mathrm{ch}_i = \mathrm{x}_i, b_1 = 1, \ldots, c_l = 0 \\ 1 & \text{if } \mathrm{ch}_i = null, \neg(b_1 = 1, \ldots, c_l = 0) \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} \Pi_k & \text{if } \mathrm{ch}_i = h_k, b_i = 1, \ldots, c_l = 0 \\ 1 - \sum_{j=1}^{n} \Pi_j & \text{if } \mathrm{ch}_i = null, b_i = 1, \ldots, c_l = 0 \\ 1 & \text{if } \mathrm{ch}_i = null, \neg(b_i = 1, \ldots, c_l = 0) \\ 0 & \text{otherwise} \end{cases}$$

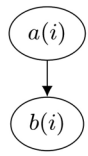
which is the same as Equation (2.7).

From Figure 2.3 and using d-separation (see Definition 17), we can see that the X_i variables are all pairwise unconditionally independent as between every couple there is the collider $\mathrm{X}_i \rightarrow \mathrm{ch}_i \leftarrow \mathrm{body}_i$.

Figure 2.4 shows $\gamma(\mathcal{P})$ for Example 28.

2.6 Generality of the Distribution Semantics

The assumption of independence of the random variables associated with ground clauses may seem restrictive. However, any probabilistic relationship between Boolean random variables that can be represented with a BN can be



$P(a(i))$

$a(i) = 0$	$1 - p_1$
$a(i) = 1$	p_1

$P(b(i)|a(i))$

$a(i)$	0	1
$b(i) = 0$	$1 - p_2$	$1 - p_3$
$b(i) = 1$	p_2	p_3

Figure 2.5 BN representing the dependency between $a(i)$ and $b(i)$.

modeled in this way. For example, suppose you want to model a general dependency between the ground atoms $a(i)$ and $b(i)$ regarding predicates $a/1$ and $b/1$ and constant i. This dependency can be represented with the BN of Figure 2.5.

The joint probability distribution $P(a(i), b(i))$ over the two Boolean random variables $a(i)$ and $b(i)$ is

$$\begin{aligned} P(0,0) &= (1-p_1)(1-p_2) \\ P(0,1) &= (1-p_1)(p_2) \\ P(1,0) &= p_1(1-p_3) \\ P(1,1) &= p_1 p_3 \end{aligned}$$

This dependency can be modeled with the following LPAD $\mathcal{P}$:

$C_1 = \quad a(i) : p_1$
$C_2 = \quad b(X) : p_2 \leftarrow a(X)$
$C_3 = \quad b(X) : p_3 \leftarrow \sim a(X)$

We can associate Boolean random variables X_1 with C_1, X_2, with $C_2\{X/i\}$, and X_3 with $C_3\{X/i\}$, where X_1, X_2, and X_3 are mutually independent. These three random variables generate eight worlds. $\neg a(i) \wedge \neg b(i)$ for example is true in the worlds

$$w_1 = \varnothing,\ w_2 = \{b(i) \leftarrow a(i)\}$$

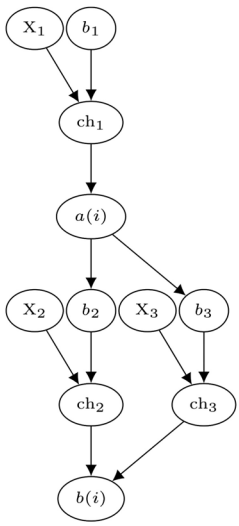
whose probabilities are

$$\begin{aligned} P'(w_1) &= (1-p_1)(1-p_2)(1-p_3) \\ P'(w_2) &= (1-p_1)(1-p_2)p_3 \end{aligned}$$

so

$$P'(\neg a(i), \neg b(i)) = (1-p_1)(1-p_2)(1-p_3) + (1-p_1)(1-p_2)p_3 = P(0,0).$$

We can prove similarly that the distributions P and P' coincide for all joint states of $a(i)$ and $b(i)$.



$P''(\mathrm{X}_1)$

$\mathrm{X}_1 = \text{null}$	$1 - p_1$
$\mathrm{X}_1 = a(i)$	p_1

$P''(\mathrm{X}_2)$

$\mathrm{X}_2 = \text{null}$	$1 - p_2$
$\mathrm{X}_2 = b(i)$	p_2

$P''(\mathrm{X}_3)$

$\mathrm{X}_3 = \text{null}$	$1 - p_3$
$\mathrm{X}_3 = b(i)$	p_3

Figure 2.6 BN modeling the distribution over $a(i)$, $b(i)$, X_1, X_2, X_3.

Modeling the dependency between $a(i)$ and $b(i)$ with the program above is equivalent to represent the BN of Figure 2.5 with the network $\gamma(\mathcal{P})$ of Figure 2.6.

Since $\gamma(\mathcal{P})$ defines the same distribution as $\mathcal{P}$, the distributions P and P'', the one defined by $\gamma(\mathcal{P})$, agree on the variables $a(i)$ and $b(i)$, i.e.,

$$P(a(i), b(i)) = P''(a(i), b(i))$$

for any value of $a(i)$ and $b(i)$. From Figure 2.6, it is also clear that X_1, X_2, and X_3 are mutually unconditionally independent, thus showing that it is possible to represent any dependency with independent random variables. So we can model general dependencies among ground atoms with the DS.

This confirms the results of Sections 2.3 and 2.5 that graphical models can be translated into probabilistic logic programs under the DS and vice versa. Therefore, the two formalisms are equally expressive.

2.7 Extensions of the Distribution Semantics

Programs under the DS may contain *flexible probabilities* [De Raedt and Kimmig, 2015] or probabilities that depend on values computed during program execution. In this case, the probabilistic annotations are variables, as in the program[12] from [De Raedt and Kimmig, 2015]

[12] http://cplint.eu/e/flexprob.pl

```
red(Prob):Prob.

draw_red(R, G):-
  Prob is R/(R + G),
  red(Prob).
```

The query `draw_red(r,g)`, where `r` and `g` are the number of green and red balls in an urn, succeeds with the same probability as that of drawing a red ball from the urn.

Flexible probabilities allow the computation of probabilities on the fly during inference. However, flexible probabilities must be ground when their value must be evaluated during inference. Many inference systems support them by imposing constraints on the form of programs.

The body of rules may also contain literals for a meta-predicate such as `prob/2` that computes the probability of an atom, thus allowing nested or meta-probability computations [De Raedt and Kimmig, 2015]. Among the possible uses of such a feature De Raedt and Kimmig [2015] mention: filtering proofs on the basis of the probability of subqueries, or implementing simple forms of combining rules.

An example of the first use is[13]

```
a:0.2:-
  prob(b,P),
  P>0.1.
```

where `a` succeeds with probability 0.2 only if the probability of `b` is larger than 0.1.

An example of the latter is[14]

```
p(P):P.

max_true(G1, G2) :-
  prob(G1, P1),
  prob(G2, P2),
  max(P1, P2, P), p(P).
```

where `max_true(G1,G2)` succeeds with the success probability of its more likely argument.

[13] http://cplint.eu/e/meta.pl

[14] http://cplint.eu/e/metacomb.pl

2.8 CP-Logic

CP-logic [Vennekens et al., 2009] is a language for representing causal laws. It shares many similarities with LPADs but specifically aims at modeling probabilistic causality. Syntactically, *CP-logic programs*, or *CP-theories*, are identical to lpads[15]: they are composed of annotated disjunctive clauses that are interpreted as follows: for each grounding

$$h_1 : \Pi_1 \ ; \ \ldots \ ; h_m : \Pi_n \leftarrow B$$

of a clause of the program, B represents an event whose effect is to cause at most one of the h_i atoms to become true and the probability of h_i of being caused is Π_i. Consider the following medical example.

Example 29 (CP-logic program – infection [Vennekens et al., 2009]). *A patient is infected by a bacterium. Infection can cause either pneumonia or angina. In turn, angina can cause pneumonia and pneumonia can cause angina. This can be represented by the CP-logic program:*

$$angina : 0.2 \leftarrow pneumonia. \tag{2.8}$$

$$pneumonia : 0.3 \leftarrow angina. \tag{2.9}$$

$$pneumonia : 0.4 \ ; \ angina : 0.1 \leftarrow infection. \tag{2.10}$$

$$infection. \tag{2.11}$$

The semantics of CP-logic programs is given in terms of probability trees that represent the possible courses of the events encoded in the program. We consider first the case where the program is positive, i.e., the bodies of rules do not contain negative literals.

Definition 18 (Probability tree – positive case). *A* probability tree[16] T *for a program* $\mathcal{P}$ *is a tree where every node* n *is labeled with a two-valued interpretation* $I(n)$ *and a probability* $P(n)$. T *is constructed as follows:*

- *The root node* r *has probability* $P(r) = 1.0$ *and interpretation* $I(r) = \varnothing$.
- *Each inner node* n *is associated with a ground clause* C_i *such that*
  - *no ancestor of* n *is associated with* C_i,
  - *all atoms in* $body(C_i)$ *are true in* $I(n)$,

[15] There are versions of CP-logic that have a more general syntax but they are not essential for the discussion here

[16] We follow here the definition of [Shterionov et al., 2015] for its simplicity.

n has one child node for each atom $h_k \in head(C_i)$. The k-th child has interpretation $I(n) \cup \{h_k\}$ and probability $P(n) \cdot \Pi_k$.

- *No leaf can be associated with a clause following the rule above.*

A probability tree defines a probability distribution $P(I)$ over the interpretation of the program $\mathcal{P}$: the probability of an interpretation I is the sum of the probabilities of the leaf nodes n such that $I = I(n)$.

The probability tree for Example 2.11 is shown in Figure 2.7. The probability distribution over the interpretations is

I	$\{inf, pn, ang\}$	$\{inf, pn\}$	$\{inf, ang\}$	$\{inf\}$
$P(I)$	0.11	0.32	0.07	0.5

There can be more than one probability tree for a program but Vennekens et al. [2009] show that all the probability trees for the program define the same probability distribution over interpretations. So we can speak of *the* probability tree for $\mathcal{P}$ and this defines the semantics of the CP-logic program. Moreover, each program has at least one probability tree.

Vennekens et al. [2009] also show that the probability distribution defined by the LPADs semantics is the same as that defined by the CP-logic semantics. So probability trees represent an alternative definition of the DS for LPADs.

If the program contains negation, checking the truth of the body of a clause must be made with care because an atom that is currently absent from $I(n)$ may become true later. Therefore, we must make sure that for each negative literal $\sim a$ in $body(C_i)$, the positive literal a cannot be made true starting from $I(n)$.

Example 30 (CP-logic program – pneumonia [Vennekens et al., 2009]). *A patient has pneumonia. Because of pneumonia, the patient is treated. If the patient has pneumonia and is not treated, he may get fever.*

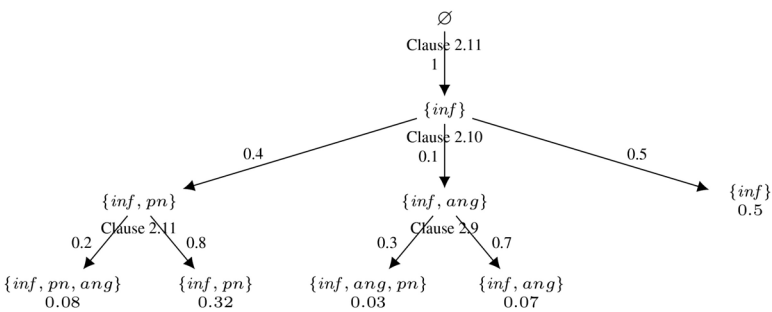


Figure 2.7 Probability tree for Example 2.11. From [Vennekens et al., 2009].

$$pneumonia. \tag{2.12}$$

$$treatment : 0.95 \leftarrow pneumonia. \tag{2.13}$$

$$fever : 0.7 \leftarrow pneumonia, \sim treatment. \tag{2.14}$$

Two probability trees for this program are shown in Figures 2.8 and 2.9. Both trees satisfy Definition 18 but define two different probability distributions. In the tree of Figure 2.8, Clause 2.14 has negative literal $\sim treatment$ *in its body and is applied at a stage where* $treatment$ *may still become true, as happens in the level below.*

In the tree of Figure 2.9, instead Clause 2.14 is applied when the only rule for $treatment$ *has already fired, so in the right child of the node at the second level* $treatment$ *will never become true and Clause 2.14 can safely be applied.*

In order to formally define this, we need the following definition that uses three-valued logic. A conjunction in three-valued logic is true or undefined if no literal in it is false.

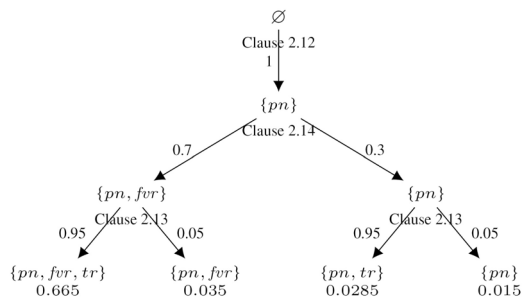


Figure 2.8 An incorrect probability tree for Example 30. From [Vennekens et al., 2009].

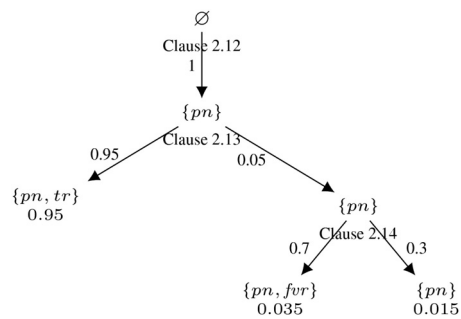


Figure 2.9 A probability tree for Example 30. From [Vennekens et al., 2009].

Definition 19 (Hypothetical derivation sequence). *A* hypothetical derivation sequence *in a node n is a sequence $(\mathcal{I}_i)_{0 \leqslant i \leqslant n}$ of three-valued interpretations that satisfy the following properties. Initially, $\mathcal{I}_0$ assigns false to all atoms not in $I(n)$. For each $i > 0$, $\mathcal{I}_{i+1} = \langle I_{T,i+1}, I_{F,i+1} \rangle$ is obtained from $\mathcal{I}_i = \langle I_{T,i}, I_{F,i} \rangle$ by considering a rule R with $body(R)$ true or undefined in $\mathcal{I}_i$ and an atom a in its head that is false in $\mathcal{I}$. Then $I_{T,i+1} = I_{T,i+1}$ and $I_{F,i+1} = I_{F,i+1} \backslash \{a\}$.*

Every hypothetical derivation sequence reaches the same limit. For a node n in a probabilistic tree, we denote this unique limit as $\mathcal{I}(n)$. It represents the set of atoms that might still become true; in other words, all the atoms in the false part of $\mathcal{I}(n)$ will never become true and so they can be considered as false.

The definition of probability tree of a program with negation becomes the following.

Definition 20 (Probability tree – general case). *A* probability tree T *for a program $\mathcal{P}$ is a tree*

- *satisfying the conditions of Definition 18, and*
- *for each node n and associated clause C_i, for each negative literal $\sim a$ in $body(C_i)$, $a \in I_F$ with $\mathcal{I}(n) = \langle I_T, I_F \rangle$.*

All the probability trees according for the program according to Definition 20 establish the same probability distribution over interpretations.

It can be shown that the set of false atoms of the limit of the hypothetical derivation sequence is equal to the greatest fixpoint of the operator $OpFalse^P_{\mathcal{I}}$ (see Definition 2) with $\mathcal{I} = \langle I(n), \varnothing \rangle$ and P a program that contains, for each rule

$$h_1 : \Pi_1 \ ; \ \ldots \ ; h_m : \Pi_n \leftarrow B$$

of $\mathcal{P}$, the rules

$$\begin{aligned} h_1 &\leftarrow B. \\ &\ldots \\ h_m &\leftarrow B. \end{aligned}$$

In other words, if $\mathcal{I}(n) = \langle I_T, I_F \rangle$ and $\text{gfp}(OpFalse^P_{\mathcal{I}}) = F$, then $I_F = F$.

In fact, for the body of a clause to be true or undefined in $\mathcal{I}_i = \langle I_{T,i}, I_{F,i} \rangle$, each positive literal a must be absent from $I_{F,i}$ and each negative literal $\sim a$ must be such that a is absent from $I_{T,i}$, which are the complementary conditions in the definition of the operator $OpFalse^P_{\mathcal{I}}(Fa)$.

On the other hand, the generation of a child n' of a node n using a rule C_i that adds an atom a to $I(n)$ can be seen as part of an application of $Op\,True^{P}_{\mathcal{I}(n)}$. So there is a strong connection between CP-logic and the WFS.

In the trees of Figures 2.8 and 2.9, the child $n = \{pn\}$ of the root has $I_F = \varnothing$, so Clause 2.14 cannot be applied as $treatment \notin I_F$ and the only tree allowed by Definition 20 is that of Figure 2.9.

The semantics of CP-logic satisfies these causality principles:

- The *principle of universal causation* states that all changes to the state of the domain must be triggered by a causal law whose precondition is satisfied.
- The *principle of sufficient causation* states that if the precondition to a causal law is satisfied, then the event that it triggers must eventually happen.

and therefore the logic is particularly suitable for representing causation.

Moreover, CP-logic satisfies the *temporal precedence assumption* that states that a rule R will not fire until its precondition is in its final state. In other words, a rule fires only when the causal process that determines whether its precondition holds is fully finished. This is enforced by the treatment of negation of CP-logic.

There are CP-logic programs that do not admit any probability tree, as the following example shows.

Example 31 (Invalid CP-logic program [Vennekens et al., 2009]). *In a two-player game, white wins if black does not win and black wins if white does not win:*

$$win(white) \leftarrow \sim win(black). \tag{2.15}$$

$$win(black) \leftarrow \sim win(white). \tag{2.16}$$

At the root of the probability tree for this program, both Clauses 2.15 and 2.16 have their body true but they cannot fire as I_F for the root is $\varnothing$. So the root is a leaf where however two rules have their body true, thus violating the condition of Definition 18 that requires that leaves cannot be associated with rules.

This theory is problematic from a causal point of view, as it is impossible to define a process that follows the causal laws. Therefore, we want to exclude these cases and consider only *valid* CP-theories.

Definition 21 (Valid CP-theory). *A CP-theory is* valid *if it has at least one probability tree.*

The equivalence of the LPADs and CP-logic semantics is also carried to the general case of programs with negation: the probability tree of a valid CP-theory defines the same distribution as that defined by interpreting the program as an LPAD.

However, there are sound LPADs that are not valid CP-theories. Recall that a sound LPAD is one where each possible world has a two-valued WFM.

Example 32 (Sound LPAD – invalid CP-theory Vennekens et al. [2009]). *Consider the program*

$p : 0.5$ *;* $q : 0.5 \leftarrow r.$

$r \leftarrow \sim p.$

$r \leftarrow \sim q.$

Such a program has no probability tree, so it is not a valid CP-theory. Its possible worlds are

$$\{p \leftarrow r; r \leftarrow \sim p; r \leftarrow \sim q\}$$

and

$$\{q \leftarrow r; r \leftarrow \sim p; r \leftarrow \sim q\}$$

that both have total WFMs, $\{r, p\}$ *and* $\{r, q\}$*, respectively, so the LPAD is sound.*

In fact, it is difficult to imagine a causal process for this program.

Therefore, LPADs and CP-logic have some differences but these arise only in corner cases, so sometimes CP-logic and LPADs are used as a synonyms. This also shows that clauses in LPADs can be assigned in many cases a causal interpretation.

The equivalence of the semantics implies that, for a valid CP-theory, the leaves of the probability tree are associated with the WFMs of the possible world obtained by considering all the clauses used in the path from the root to the leaf with the head selected according to the choice of child. If the program is deterministic, the only leaf is associated with the total-well founded model of the program.

2.9 Semantics for Non-Sound Programs

In Section 2.2, we considered only sound programs, those for which every world has a two-valued WFM. In this way, we avoid non-monotonic aspects of the program and we deal with uncertainty only by means of probability theory.

When a program is not sound in fact, assigning a semantics to probabilistic logic programs is not obvious, as the next example shows.

Example 33 (Insomnia [Cozman and Mauá, 2017]). *Consider the program*

$sleep \leftarrow \sim work, \sim insomnia.$
$work \leftarrow \sim sleep.$
$\alpha :: insomnia.$

This program has two worlds, w_1 containing $insomnia$ and w_2 not containing it. The first has the single stable model and total WFM

$$I_1 = \{insomnia, \sim sleep, \sim work\}$$

The latter has two stable models

$$I_2 = \{insomnia, \sim sleep, work\}$$
$$I_3 = \{insomnia, sleep, \sim work\}$$

and a WFM $\mathcal{I}_2$ where $insomnia$ is true and the other two atoms are undefined.

If we ask for the probability of $sleep$, the first world, w_1, with probability α, surely doesn't contribute. We are not sure instead what to do with the second, as $sleep$ is included in only one of the two stable models and it is undefined in the WFM.

To handle programs like the above, Hadjichristodoulou and Warren [2012] proposed the WFS for probabilistic logic programs where a program defines a probability distribution over WFMs rather than two-valued models. This induces a probability distribution over random variables associated with atoms that are, however, three-valued instead of Boolean.

An alternative approach, the *credal semantics* [Cozman and Mauá, 2017], sees such programs as defining a set of probability measures over the interpretations. The name derives from the fact that sets of probability distributions are often called *credal sets*.

The semantics considers programs syntactically equal to ProbLog (i.e., non-probabilistic rules and probabilistic facts) and generates worlds as in ProbLog. The semantics requires that each world of the program has at least one stable models. Such programs are called *consistent*.

A program then defines a set of probability distributions over the set of all possible two-valued interpretations of the program. Each distribution P in the set is called a *probability model* and must satisfy two conditions:

1. every interpretation I for which $P(I) > 0$ must be a stable model of the world w_σ that agrees with I on the truth value of the probabilistic facts;

2. the sum of the probabilities of the stable models of w must be equal to $P(\sigma)$.

A set of distributions is obtained because we do not fix how the probability mass $P(\sigma)$ of a world w_σ is distributed over its stable models when there is more than one. We indicate with $\mathbf{P}$ the set of probability models and call it the *credal semantics* of the program. Given a probability model, we can compute the probability of a query q as for the Distribution Semantics (DS), by summing $P(I)$ for all the interpretations I where q is true.

In this case, given a query q, we are interested in the *lower and upper probabilities* of q defined as

$$\underline{P}(q) = \inf_{P \in \mathbf{P}} P(q)$$
$$\overline{P}(q) = \sup_{P \in \mathbf{P}} P(q)$$

If we are also given evidence e, Cozman and Mauá [2017] define *lower and upper conditional probabilities* as

$$\underline{P}(q|e) = \inf_{P \in \mathbf{P}, P(e)>0} P(q)$$
$$\overline{P}(q|e) = \sup_{P \in \mathbf{P}, P(e)>0} P(q)$$

and leave them undefined when $P(e) = 0$ for all $P \in \mathbf{P}$.

Example 34 (Insomnia – continued – [Cozman and Mauá, 2017]). *Consider again the program of Example 33. A probability model that assigns the following probabilities to the models of the program*

$$P(I_1) = \alpha$$
$$P(I_2) = \gamma(1-\alpha)$$
$$P(I_3) = (1-\gamma)(1-\alpha)$$

for $\gamma \in [0, 1]$, satisfies the two conditions of the semantics, and thus belongs to $\mathbf{P}$. The elements of $\mathbf{P}$ are obtained by varying γ.

Considering the query $sleep$, we can easily see that $\underline{P(sleep = true)} = 0$ and $\overline{P(sleep = true)} = 1 - \alpha$.

With the semantics of [Hadjichristodoulou and Warren, 2012] instead, we have

$$P(I_1) = \alpha$$
$$P(\mathcal{I}_2) = 1 - \alpha$$

so

$$P(sleep = true) = 0$$
$$P(sleep = false) = \alpha$$
$$P(sleep = undefined) = 1 - \alpha.$$

Example 35 (Barber paradox – [Cozman and Mauá, 2017]). *The barber paradox was introduced by Russell [1967]. If the village barber shaves all, and only, those in the village who don't shave themselves, does the barber shave himself?*

A probabilistic version of this paradox can be encoded with the program

$shaves(X, Y) \leftarrow barber(X), villager(Y), \sim shaves(Y, Y).$
$villager(a).$
$barber(b).$
$0.5 :: villager(b).$

and the query $shaves(b, b)$.

The program has two worlds, w_1 *and* w_2*, the first not containing the fact* $villager(b)$ *and the latter containing it. The first world has a single stable model* $I_1 = \{villager(a), barber(b), shaves(b, a)\}$ *that is also the total WFM. In the latter world, the rule has an instance that can be simplified to* $shaves(b, b) \leftarrow\sim shaves(b, b)$*. Since it contains a loop through an odd number of negations, the world has no stable model and the three-valued WFM:*

$$\mathcal{I}_2 = \{villager(a), barber(b), shaves(b, a), \sim shaves(a, a), \sim shaves(a, b)\}.$$

So the program is not consistent and the credal semantics is not defined for it, while the semantics of [Hadjichristodoulou and Warren, 2012] is still defined and would yield

$$P(shaves(b, b) = true) = 0.5$$
$$P(shaves(b, b) = undefined) = 0.5$$

The WFS for probabilistic logic programs assigns a semantics to more programs. However, it introduces the truth value *undefined* that expresses uncertainty and, since probability is used as well to deal with uncertainty, some confusion may arise. For example, one may ask what is the value of $(q = true|e = undefined)$. If $e = undefined$ means that we don't know anything about e, then $P(q = true|e = undefined)$ should be equal to $P(q = true)$ but this is not true in general. The credal semantics avoids these problems by considering only two truth values.

Cozman and Mauá [2017] show that the set $\mathbf{P}$ is the set of all probability measures that dominate an infinitely monotone Choquet capacity.

An *infinitely monotone Choquet capacity* is a function $\underline{P}$ from an algebra Ω on a set W to the real interval $[0, 1]$ such that

1. $\underline{P}(W) = 1 - \underline{P}(\varnothing) = 1$, and
2. for any $\omega_1, \ldots, \omega_n \subseteq \Omega$,

$$\underline{P}(\cup_i \omega_i) \geqslant \sum_{J \subseteq \{1,\ldots,n\}} (-1)^{|J|+1} \underline{P}(\cap_{j \in J} \omega_j) \tag{2.17}$$

Infinitely monotone Choquet capacity is a generalization of finitely additive probability measures: the latter are special cases of the first where Equation (2.17) holds with equality. In fact, the right member of Equation (2.17) is an application of the inclusion–exclusion principle that gives the probability of the union of non-disjoint sets. Infinitely monotone Choquet capacities also appear as belief functions of Dempster–Shafer theory [Shafer, 1976].

Given an infinitely monotone Choquet capacity $\underline{P}$, we can construct the set of measures $D(\underline{P})$ that dominate $\underline{P}$ as

$$D(\underline{P}) = \{P | \forall \omega \in \Omega : P(\omega) \geqslant \underline{P}(\omega)\}$$

We say that $\underline{P}$ *generates* the credal set $D(\underline{P})$ and we call $D(\underline{P})$ an *infinitely monoton credal set*. It is possible to show that the lower probability of $D(\underline{P})$ is exactly the generating infinitely monotone Choquet capacity: $\underline{P}(\omega) = \inf_{P \in D(\underline{P})} P(\omega)$.

Infinitely monotone credal sets are closed and convex. Convexity here means that if P_1 and P_2 are in the credal set, then $\alpha P_1 + (1 - \alpha) P_2$ is also in the credal set for $\alpha \in [0, 1]$. Given a consistent program, its credal semantics is thus a closed and convex set of probability measures.

Moreover, given a query q, we have

$$\underline{P}(q) = \sum_{w \in W, AS(w) \subseteq J_q} P(\sigma) \quad \overline{P}(q) = \sum_{w \in W, AS(w) \cap J_q \neq \varnothing} P(\sigma)$$

where J_q is the set of interpretations where q is true and $AS(w)$ is the set of stable models of world w_σ.

The lower and upper conditional probabilities of a query q are given by:

$$\underline{P}(q|e) = \frac{\underline{P}(q, e)}{\underline{P}(q, e) + \overline{P}(\neg q, e)} \tag{2.18}$$

$$\overline{P}(q|e) = \frac{\overline{P}(q, e)}{\overline{P}(q, e) + \underline{P}(\neg q, e)} \tag{2.19}$$

2.10 KBMC Probabilistic Logic Programming Languages

In this section, we present three examples of KBMC languages: Bayesian Logic Programs (BLPs), CLP(BN), and the Prolog Factor Language (PFL).

2.10.1 Bayesian Logic Programs

BLPs [Kersting and De Raedt, 2001] use logic programming to compactly encode a large BN. In BLPs, each ground atom represents a (not necessarily Boolean) random variable and the clauses define the dependencies between ground atoms. A clause of the form

$$a|a_1, \ldots, a_m$$

indicates that, for each of its groundings $(a|a_1, \ldots, a_m)\theta$, $a\theta$ has $a_1\theta$, ..., $a_m\theta$ as parents. The domains and CPTs for the ground atom/random variables are defined in a separate portion of the model. In the case where a ground atom $a\theta$ appears in the head of more than one clause, a *combining rule* is used to obtain the overall CPT from those given by individual clauses.

For example, in the Mendelian genetics program of Example 25, the dependency that gives the value of the color gene on chromosome 1 of a plant as a function of the color genes of its mother can be expressed as

cg(X,1)|mother(Y,X),cg(Y,1),cg(Y,2).

where the domain of atoms built on predicate *cg/2* is $\{p,w\}$ and the domain of *mother(Y,X)* is Boolean. A suitable CPT should then be defined that assigns equal probability to the alleles of the mother to be inherited by the plant.

Various learning systems use BLPs as the representation language: RBLP [Revoredo and Zaverucha, 2002; Paes et al., 2005], PFORTE [Paes et al., 2006], and SCOOBY [Kersting and De Raedt, 2008].

2.10.2 CLP(BN)

In a CLP(BN) program [Costa et al., 2003], logical variables can be random. Their domain, parents, and CPTs are defined by the program. Probabilistic dependencies are expressed by means of constraints as in Constraint Logic Programming (CLP):

```
{ Var = Function with p(Values, Dist) }
{ Var = Function with p(Values, Dist, Parents) }
```

The first form indicates that the logical variable `Var` is random with domain `Values` and CPT `Dist` but without parents; the second form defines a random variable with parents. In both forms, `Function` is a term over logical variables that is used to parameterize the random variable: a different random variable is defined for each instantiation of the logical variables in the term. For example, the following snippet from a school domain:

```
course_difficulty(CKey, Dif) :-
  { Dif = difficulty(CKey) with p([h,m,l],
    [0.25, 0.50, 0.25]) }.
```

defines the random variable `Dif` with values `h`, `m`, and `l` representing the difficulty of the course identified by `CKey`. There is a different random variable for every instantiation of `CKey`, i.e., for each course. In a similar manner, the intelligence `Int` of a student identified by `SKey` is given by

```
student_intelligence(SKey, Int) :-
  { Int = intelligence(SKey) with p([h, m, l],
    [0.5,0.4,0.1]) }.
```

Using the above predicates, the following snippet predicts the grade received by a student when taking the exam of a course.

```
registration_grade(Key, Grade) :-
  registration(Key, CKey, SKey),
  course_difficulty(CKey, Dif),
  student_intelligence(SKey, Int),
  { Grade = grade(Key) with  p(['A','B','C','D'],
   % h/h  h/m  h/l  m/h  m/m  m/l  l/h  l/m  l/l
    [0.20,0.70,0.85,0.10,0.20,0.50,0.01,0.05,0.10,
     % 'A'
     0.60,0.25,0.12,0.30,0.60,0.35,0.04,0.15,0.40,
     % 'B'
     0.15,0.04,0.02,0.40,0.15,0.12,0.50,0.60,0.40,
     % 'C'
     0.05,0.01,0.01,0.20,0.05,0.03,0.45,0.20,0.10],
     % 'D'
   [Int,Dif]) }.
```

Here `Grade` indicates a random variable parameterized by the identifier `Key` of a registration of a student to a course. The code states that there

is a different random variable `Grade` for each student's registration in a course and each such random variable has possible values ``'A'', ``'B'', ``'C'' and ``'D''. The actual value of the random variable depends on the intelligence of the student and on the difficulty of the course, that are thus its parents. Together with facts for `registration/3` such as

```
registration(r0,c16,s0).  registration(r1,c10,s0).
registration(r2,c57,s0).  registration(r3,c22,s1).
....
```

the code defines a BN with a `Grade` random variable for each registration. CLP(BN) is implemented as a library of YAP Prolog. The library performs query answering by constructing the sub-network that is relevant to the query and then applying a BN inference algorithm.

The unconditional probability of a random variable can be computed by simply asking a query to the YAP command line.

The answer will be a probability distribution over the values of the logical variables of the query that represent random variables, as in

```
?- registration_grade(r0,G).
      p(G=a)=0.4115,
      p(G=b)=0.356,
      p(G=c)=0.16575,
      p(G=d)=0.06675 ?
```

Conditional queries can be posed by including in the query ground atoms representing the evidence.

For example, the probability distribution of the grades of registration `r0` given that the intelligence of the student is high (`h`) is given by

```
?- registration_grade(r0,G),
      student_intelligence(s0,h).
      p(G=a)=0.6125,
      p(G=b)=0.305,
      p(G=c)=0.0625,
      p(G=d)=0.02 ?
```

In general, CLP provides a useful tool for Probabilistic Logic Programming (PLP), as is testified by the proposals clp(pdf(Y)) [Angelopoulos, 2003, 2004] and Probabilistic Constraint Logic Programming (PCLP) [Michels et al., 2015], see Section 4.5.

2.10.3 The Prolog Factor Language

The PFL [Gomes and Costa, 2012] is an extension of Prolog for representing first-order probabilistic models.

Most graphical models such as BNs and MNs concisely represent a joint distribution by encoding it as a set of factors. The probability of a set of variables $\mathbf{X}$ taking value $\mathbf{x}$ can be expressed as the product of n factors as:

$$P(\mathbf{X} = \mathbf{x}) = \frac{\prod_{i=1,\ldots,n} \phi_i(\mathbf{x}_i)}{Z}$$

where $\mathbf{x}_i$ is a sub-vector of $\mathbf{x}$ on which the i-th factor depends and Z is the normalization constant. Often, in a graphical model, the same factors appear repeatedly in the network, and thus we can parameterize these factors in order to simplify the representation.

A Parameterized Random Variables (PRVs) is a logical atom representing a set of random variables, one for each of its possible ground instantiations. We indicate PRV as $\mathsf{X}, \mathsf{Y}, \ldots$ and vectors of PRVs as $\mathbf{X}, \mathbf{Y}, \ldots$

A *parametric factor* or *parfactor* [Kisynski and Poole, 2009b] is a triple $\langle \mathcal{C}, \mathcal{V}, F \rangle$ where $\mathcal{C}$ is a set of inequality constraints on parameters (logical variables), $\mathcal{V}$ is a set of PRVs and F is a factor that is a function from the Cartesian product of ranges of PRVs in $\mathcal{V}$ to real values. A parfactor is also represented as $F(\mathcal{V})|\mathcal{C}$ or $F(\mathcal{V})$ if there are no constraints. A constrained PRV is of the form $\mathsf{V}|\mathcal{C}$, where $\mathsf{V} = p(X_1, \ldots, X_n)$ is a non-ground atom and $\mathcal{C}$ is a set of constraints on logical variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$. Each constrained PRV represents the set of random variables $\{P(\boldsymbol{x})|\boldsymbol{x} \in \mathcal{C}\}$, where $\boldsymbol{x}$ is the tuple of constants $(x_1, \ldots, x_n)$. Given a (constrained) PRV V, we use $RV(\mathsf{V})$ to denote the set of random variables it represents. Each ground atom is associated with one random variable, which can take any value in $range(\mathsf{V})$.

The PFL extends Prolog to support probabilistic reasoning with parametric factors. A PFL factor is a parfactor of the form

$$Type\ \mathbf{F}\ ;\ \phi\ ;\ \mathcal{C},$$

where $Type$ refers to the type of the network over which the parfactor is defined ($bayes$ for directed networks or $markov$ for undirected ones); $\mathbf{F}$ is a sequence of Prolog goals each defining a PRV under the constraints in $\mathcal{C}$ (the arguments of the factor). If $\boldsymbol{L}$ is the set of all logical variables in $\mathbf{F}$, then $\mathcal{C}$ is a list of Prolog goals that impose bindings on $\boldsymbol{L}$ (the successful substitutions for

the goals in $\mathcal{C}$ are the valid values for the variables in $\boldsymbol{L}$). ϕ is the table defining the factor in the form of a list of real values. By default, all random variables are Boolean but a different domain may be defined. Each parfactor represents the set of its groundings. To ground a parfactor, all variables of $\boldsymbol{L}$ are replaced with the values permitted by constraints in $\mathcal{C}$. The set of ground factors defines a factorization of the joint probability distribution over all random variables.

Example 36 (PFL program). *The following PFL program is inspired by the workshop attributes problem of [Milch et al., 2008]. It models the organization of a workshop where a number of people have been invited.* `series` *indicates whether the workshop is successful enough to start a series of related meetings while* `attends(P)` *indicates whether person* `P` *attends the workshop.*

This problem can be modeled by a PFL program such as

```
bayes series, attends(P); [0.51, 0.49, 0.49, 0.51];
  [person(P)].
bayes attends(P), at(P,A); [0.7, 0.3, 0.3, 0.7];
  [person(P),attribute(A)].
```

A workshop becomes a series because people attend. People attend the workshop depending on the workshop's attributes such as location, date, fame of the organizers, etc. The probabilistic atom `at(P,A)` *represents whether person* `P` *attends because of attribute* `A`.

The first PFL factor has the random variables `series` *and* `attends(P)` *as arguments (both Boolean),* `[0.51,0.49,0.49,0.51]` *as table and the list* `[person(P)]` *as constraint.*

Since KBMC languages are defined on the basis of a translation to graphical models, translations can be built between PLP languages under the DS and KBMC languages. The first have the advantage that they have a semantics that can be understood in logical terms, without necessarily referring to an underlying graphical model.

2.11 Other Semantics for Probabilistic Logic Programming

Here we briefly discuss a few examples of PLP frameworks that don't follow the distribution semantics. Our goal in this section is simply to give the flavor of other possible approaches; a complete account of such frameworks is beyond the scope of this book.

2.11.1 Stochastic Logic Programs

Stochastic Logic Programs (SLPs) [Muggleton et al., 1996; Cussens, 2001] are logic programs with parameterized clauses which define a distribution over refutations of goals. The distribution provides, by marginalization, a distribution over variable bindings for the query. SLPs are a generalization of stochastic grammars and hidden Markov models.

An *SLP* S is a definite logic program where some of the clauses are of the form $p : C$ where $p \in \mathbb{R}, p \geqslant 0$, and C is a definite clause. Let $n(S)$ be the definite logic program obtained by removing the probability labels. A *pure* SLP is an SLP where all clauses have probability labels. A *normalized* SLP is one where probability labels for clauses whose heads share the same predicate symbol sum to one.

In pure SLPs, each SLD derivation for a query q is assigned a real label by multiplying the labels of each individual derivation step. The label of a derivation step where the selected atom unifies with the head of clause $p_i : C_i$ is p_i. The probability of a successful derivation from q is the label of the derivation divided by the sum of the labels of all the successful derivations. This forms a distribution over successful derivations from q.

The probability of an instantiation $q\theta$ is the sum of the probabilities of the successful derivations that produce $q\theta$. It can be shown that the probabilities of all the atoms for a predicate q that succeed in $n(S)$ sum to one, i.e., S defines a probability distribution over the success set of q in $n(S)$.

In impure SLPs, the unparameterized clauses are seen as non-probabilistic domain knowledge acting as constraints. Derivations are identified with the set of the parameterized clauses they use. In this way, derivations that differ only on the unparameterized clauses form an equivalence class.

In practice, SLPs define probability distributions over the children of nodes of the SLD tree for a query: a derivation step $u \rightarrow v$ that connects node u with child node v is assigned a probability $P(v|u)$. This induces a probability distributions over paths from the root to the leaves of the SLD tree and in turn over answers for the query.

Given their similarity with stochastic grammars and hidden Markov models, SLPs are particularly suitable for representing these kinds of models. They differ from the DS because they define a probability distribution over instantiations of the query, while the DS usually defines a distribution over the truth values of ground atoms.

Example 37 (Probabilistic context-free grammar – SLP). *Consider the probabilistic context free grammar:*

$0.2 : S \rightarrow aS$
$0.2 : S \rightarrow bS$
$0.3 : S \rightarrow a$
$0.3 : S \rightarrow b$

The SLP

$0.2 : s([a|R]) \leftarrow s(R).$
$0.2 : s([b|R]) \leftarrow s(R).$
$0.3 : s([a]).$
$0.3 : s([b]).$

defines a distribution over the values of S *in* $s(S)$ *that is the same as the one defined by the probabilistic context-free grammar above. For example,* $P(s([a,b])) = 0.2 \cdot 0.3 = 0.6$ *according to the program and* $P(ab) = 0.2 \cdot 0.3 = 0.6$ *according to the grammar.*

Various approaches have been proposed for learning SLPs. Muggleton [2000a,b] proposed to use an Inductive Logic Programming (ILP) system, Progol [Muggleton, 1995], for learning the structure of the programs, and a second phase where the parameters are tuned using a generalization of relative frequency.

Parameters are also learned by means of optimization in failure-adjusted maximization [Cussens, 2001; Angelopoulos, 2016] and by solving algebraic equations [Muggleton, 2003].

2.11.2 ProPPR

ProPPR [Wang et al., 2015] is an extension of SLPs that that is related to Personalized PageRank (PPR) [Page et al., 1999].

ProPPR extends SLPs in two ways. The first is the method for computing the labels of the derivation steps. A derivation step $u \rightarrow v$ is not simply assigned the parameter associated with the clause used in the step. Instead, the label of the derivation step, $P(v|u)$ is computed using a log-linear model $P(v|u) \propto \exp(\mathbf{w} \cdot \phi_{u \rightarrow v})$ where $\mathbf{w}$ is a vector of real-valued weights and $\phi_{u \rightarrow v}$ is a 0/1 vector of "features" that depend on the clause being used. The features are user defined and the association between clauses and features is indicated using annotations.

Example 38 (ProPPR program). *The ProPPR program [Wang et al., 2015]*

$about(X, Z) \leftarrow handLabeled(X, Z). \quad \#base$
$about(X, Z) \leftarrow sim(X, Y), about(Y, Z). \quad \#prop$

$sim(X,Y) \leftarrow link(X,Y).$ $\#sim, link$
$sim(X,Y) \leftarrow hasWord(X,W), hasWord(Y,W),$
$\quad linkedBy(X,Y,W).$ $\#sim, word$
$linkedBy(X,Y,W).$ $\#by(W)$

can be used to compute the topic of web pages on the basis of possible hand labeling or similarity with other web pages. Similarity is defined as well in a probabilistic way depending on the links and words between the two pages.

Clauses are annotated with a list of atoms (indicated after the # symbol) that may contain variables from the head of clauses. In the example, the third clause is annotated with the list of atoms $sim, link$ while the last clause is annotated by the atom $by(W)$. Each grounding of each atom in the list stands for a different feature, so for example sim, $link$, and $by(sprinter)$ stand for three different features. The vector $\phi_{u \to v}$ is obtained by assigning value 1 to the features associated with the atoms in the annotation of the clause used for the derivation step $u \to v$ and value 0 otherwise. If the atoms contain variables, these are shared with the head of the clause and are grounded with the values of the clause instantiation used in $u \to v$.

So a ProPPR program is defined by an annotated program plus values for the weights $\mathbf{w}$. This annotation approach considerably increases the flexibility of SLP labels: ProPPR annotations can be shared across clauses and can yield labels that depend on the particular clause grounding that is used by the derivation step. An SLP is a ProPPR program where each clause has a different annotation consisting of an atom without arguments.

The second way in which ProPPR extend SLPs consists in the addition of edges to the SLD tree: an edge is added (a) from every solution leaf to itself; and (b) from every node to the start node.

The procedure for assigning probabilities to queries of SLP can then be applied to the resulting graph. The self-loop links heuristically upweight solution nodes and the restart links make SLP's graph traversal a PPR procedure [Page et al., 1999]: a PageRank can be associated with each node, representing the probability that a random walker starting from the root arrives in that node.

The restart links favor the results of short proofs: if the restart probability is α for every node u, then the probability of reaching any node at depth d is bounded by $(1-\alpha)^d$.

Parameter learning for ProPPR is performed in [Wang et al., 2015] by stochastic gradient descent.

2.12 Other Semantics for Probabilistic Logics

In this section, we discuss semantics for probabilistic logic languages that are not based on logic programming.

2.12.1 Nilsson's Probabilistic Logic

Nilsson's probabilistic logic [Nilsson, 1986] takes an approach for combining logic and probability that is different from the DS: while the first considers sets of distributions, the latter computes a single distribution over possible worlds. In Nilsson's logic, a *probabilistic interpretation* Pr defines a probability distribution over the set of interpretations $Int2$. The *probability of a logical formula* F according to Pr, denoted $Pr(F)$, is the sum of all $Pr(I)$ such that $I \in Int2$ and $I \models F$. A *probabilistic knowledge base* $\mathcal{K}$ is a set of probabilistic formulas of the form $F \geqslant p$. A probabilistic interpretation Pr *satisfies* $F \geqslant p$ iff $Pr(F) \geqslant p$. Pr *satisfies* $\mathcal{K}$, or Pr is a *model* of $\mathcal{K}$, iff Pr satisfies all $F \geqslant p \in \mathcal{K}$. $Pr(F) \geqslant p$ is a *tight logical consequence* of $\mathcal{K}$ iff p is the infimum of $Pr(F)$ in the set of all models Pr of $\mathcal{K}$. Computing tight logical consequences from probabilistic knowledge bases can be done by solving a linear optimization problem.

With Nilsson's logic, the consequences that can be obtained from logical formulas differ from those of the DS. Consider a ProbLog program (see Section 2.1) composed of the facts $0.4 :: c(a)$ and $0.5 :: c(b)$, and a probabilistic knowledge base composed of $c(a) \geqslant 0.4$ and $c(b) \geqslant 0.5$. For the DS, $P(c(a) \vee c(b)) = 0.7$, while with Nilsson's logic, the lowest p such that $Pr(c(a) \vee c(b)) \geqslant p$ holds is 0.5. This difference is due to the fact that, while Nilsson's logic makes no assumption about the independence of the statements, in the DS, the probabilistic axioms are considered as independent. While independencies can be encoded in Nilsson's logic by carefully choosing the values of the parameters, reading off the independencies from the theories becomes more difficult.

However, the assumption of independence of probabilistic axioms does not restrict expressiveness as shown in Section 2.6.

2.12.2 Markov Logic Networks

A Markov Logic Network (MLN) is a first-order logical theory in which each sentence is associated with a real-valued weight. An MLN is a template for generating MNs. Given sets of constants defining the domains of the logical variables, an MLN defines an MN that has a Boolean node for each ground

atom and edges connecting the atoms appearing together in a grounding of a formula. MLNs follow the KBMC approach for defining a probabilistic model [Wellman et al., 1992; Bacchus, 1993]. The probability distribution encoded by an Markov Logic Network (MLN) is

$$P(\mathbf{x}) = \frac{1}{Z} \exp(\sum_{f_i \in M} w_i n_i(\mathbf{x}))$$

where $\mathbf{x}$ is a joint assignment of truth value to all atoms in the Herbrand base (finite because of no function symbols), M is the MLN, f_i is the i-th formula in M, w_i is its weight, $n_i(\mathbf{x})$ is the number of groundings of formula f_i that are satisfied in $\mathbf{x}$, and Z is a normalization constant.

Example 39 (Markov Logic Network). *The following MLN encodes a theory on the intelligence of friends and on the marks people get:*

```
1.5 Intelligent(x) => GoodMarks(x)
1.1 Friends(x,y) => (Intelligent(x)<=>
                     Intelligent(y))
```

The first formula gives a positive weight to the fact that if someone is intelligent, then he gets good marks in the exams he takes. The second formula gives a positive weight to the fact that friends have similar intelligence: in particular, the formula states that if `x` *and* `y` *are friends, then* `x` *is intelligent if and only if* `y` *is intelligent, so they are either both intelligent or both not intelligent.*

If the domain contains two individuals, Anna and Bob, indicated with A and B, we get the ground MN of Figure 2.10.

2.12.2.1 Encoding Markov Logic Networks with Probabilistic Logic Programming

It is possible to encode MNs and MLNs with LPADs. The encoding is based on the BN that is equivalent to the MN as discussed in Section 1.6: an MN

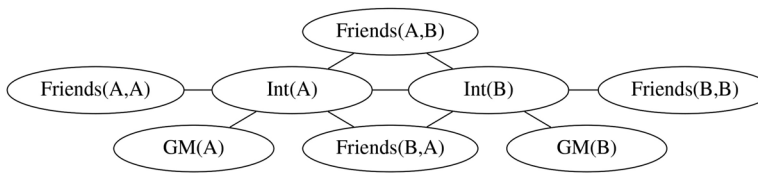


Figure 2.10 Ground Markov network for the MLN of Example 39.

factor can be represented with an extra node in the equivalent BN that is always observed. In order to model MLN formulas with LPADs, we can add an extra atom $clause_i(\boldsymbol{X})$ for each formula $F_i = w_i\ C_i$ where w_i is the weight associated with C_i and $\boldsymbol{X}$ is the vector of variables appearing in C_i. Then, when we ask for the probability of query q given evidence e, we have to ask for the probability of q given $e \wedge ce$, where ce is the conjunction of the groundings of $clause_i(\boldsymbol{X})$ for all values of i.

Clause C_i must be transformed into a Disjunctive Normal Form (DNF) formula $C_{i1} \vee \ldots \vee C_{in_i}$, where the disjuncts are mutually exclusive and the LPAD should contain the clauses

$$clause_i(\boldsymbol{X}) : e^\alpha/(1+e^\alpha) \leftarrow C_{ij}$$

for all j in $1, ..., n_i$, where $1 + e^\alpha \geqslant \max_{\mathbf{x_i}} \phi(\mathbf{x}_i) = \max\{1, e^\alpha\}$. Similarly, $\neg C_i$ must be transformed into a DNF $D_{i1} \vee \ldots \vee D_{im_i}$ and the LPAD should contain the clauses

$$clause_i(\boldsymbol{X}) : 1/(1+e^\alpha) \leftarrow D_{il}$$

for all l in $1, ..., m_i$.

Moreover, for each predicate p/n, we should add the clause

$$p(\boldsymbol{X}) : 0.5.$$

to the program, assigning *a priori* uniform probability to every ground atom.

Alternatively, if α is negative, e^α will be smaller than 1 and $\max_{\mathbf{x_i}} \phi(\mathbf{x}_i) = 1$. So we can use the two probability values e^α and 1 with the clauses

$$\begin{aligned} clause_i(\boldsymbol{X}) &: e^\alpha \leftarrow C_{ij}. \\ clause_i(\boldsymbol{X}) &\quad\ \leftarrow D_{il}. \end{aligned}$$

This solution has the advantage that some clauses are non-probabilistic, reducing the number of random variables. If α is positive in the formula $\alpha\ C$, we can consider the equivalent formula $-\alpha\ \neg C$.

The transformation above is illustrated by the following example. Given the MLN

```
1.5 Intelligent(x) => GoodMarks(x)
1.1 Friends(x,y) => (Intelligent(x)<=>Intelligent(y))
```

the first formula is translated to the clauses:

```
clause1(X):0.8175 :- \+intelligent(X).
clause1(X):0.1824 :- intelligent(X),
                     \+good_marks(X).
clause1(X):0.8175 :- intelligent(X),good_marks(X).
```

where $0.8175 = e^{1.5}/(1 + e^{-1.5})$ and $0.1824 = 1/(1 + e^{-1.5})$.
The second formula is translated to the clauses

```
clause2(X,Y):0.7502 :- \+friends(X,Y).
clause2(X,Y):0.7502 :- friends(X,Y),
                       intelligent(X),
                       intelligent(Y).
clause2(X,Y):0.7502 :- friends(X,Y),
                       \+intelligent(X),
                       \+intelligent(Y).
clause2(X,Y):0.2497 :- friends(X,Y),
                       intelligent(X),
                       \+intelligent(Y).
clause2(X,Y):0.2497 :- friends(X,Y),
                       \+intelligent(X),
                       intelligent(Y).
```

where $0.7502 = e^{1.1}/(1 + e^{1.1})$ and $0.2497 = 1/(1 + e^{1.1})$.

A priori we have a uniform distribution over student intelligence, good marks, and friendship:

```
intelligent(_):0.5.
good_marks(_):0.5.
friends(_,_):0.5.
```

and there are two students:

```
student(anna).
student(bob).
```

We have evidence that Anna is friend with Bob and Bob is intelligent. The evidence must also include the truth of all groundings of the $clause_i$ predicates:

```
evidence_mln :- clause1(anna),clause1(bob),
     clause2(anna,anna),clause2(anna,bob),
     clause2(bob,anna),clause2(bob,bob).
ev_intelligent_bob_friends_anna_bob :-
     intelligent(bob),friends(anna,bob),
     evidence_mln.
```

The probability that Anna gets good marks given the evidence is thus

$$P(\texttt{good\_marks(anna)}|\texttt{ev\_intelligent\_bob\_friends\_anna\_bob})$$

while the prior probability of Anna getting good marks is given by

$$P(\texttt{good\_marks(anna)}).$$

The probability resulting from the first query is higher ($P = 0.733$) than the second query ($P = 0.607$), since it is conditioned to the evidence that Bob is intelligent and Anna is his friend.

In the alternative transformation, the first MLN formula is translated to:

```
clause1(X) :- \+intelligent(X).
clause1(X):0.2231 :- intelligent(X),\+good_marks(X).
clause1(X) :- intelligent(X), good_marks(X).
```

where $0.2231 = e^{-1.5}$.

MLN formulas can also be added to a regular probabilistic logic program. In this case, their effect is equivalent to a soft form of evidence, where certain worlds are weighted more than others. This is the same as soft evidence in Figaro [Pfeffer, 2016]. MLN hard constraints, i.e., formulas with an infinite weight, can instead be used to rule out completely certain worlds, those violating the constraint. For example, given hard constraint C equivalent to the disjunction $C_{i1} \vee \ldots \vee C_{in_i}$, the LPAD should contain the clauses

$$clause_i(\boldsymbol{X}) \leftarrow C_{ij}$$

for all j, and the evidence should contain $clause_i(\boldsymbol{x})$ for all groundings $\boldsymbol{x}$ of $\boldsymbol{X}$. In this way, the worlds that violate C are ruled out.

2.12.3 Annotated Probabilistic Logic Programs

In Annotated Probabilistic Logic Programming (APLP) [Ng and Subrahmanian, 1992], program atoms are annotated with intervals that can be interpreted probabilistically. An example rule in this approach is:

$$a : [0.75, 0.85] \leftarrow b : [1, 1], c : [0.5, 0.75]$$

that states that the probability of a is between 0.75 and 0.85 if b is certainly true and the probability of c is between 0.5 and 0.75. The probability interval of a conjunction or disjunction of atoms is defined using a *combinator* to

construct the tightest bounds for the formula. For instance, if d is annotated with $[l_d, h_d]$ and e with $[l_e, h_e]$, the probability of $e \wedge d$ is annotated with

$$[max(0, l_d + l_e - 1), min(h_d, h_e)].$$

Using these combinators, an inference operator and fixpoint semantics is defined for positive Datalog programs. A model theory is obtained for such programs by considering the annotations as constraints on acceptable probabilistic worlds: an APLP thus describes a family of probabilistic worlds.

APLPs have the advantage that deduction is of low complexity, as the logic is truth-functional, i.e., the probability of a query can be computed directly using combinators. The corresponding disadvantages are that APLPs may be inconsistent if they are not carefully written, and that the use of the above combinators may quickly lead to assigning overly slack probability intervals to certain atoms. These aspects are partially addressed by hybrid APLPs Dekhtyar and Subrahmanian [2000], which allow different flavors of combinators based on, e.g., independence or mutual exclusivity of given atoms.