Weighted Rules under the Stable Model Semantics

Joohyung Lee and Yi Wang
School of Computing, Informatics and Decision Systems Engineering
Arizona State University, Tempe, USA
{joolee, ywang485}@asu.edu

Abstract

We introduce the concept of weighted rules under the stable model semantics following the log-linear models of Markov Logic. This provides versatile methods to overcome the deterministic nature of the stable model semantics, such as resolving inconsistencies in answer set programs, ranking stable models, associating probability to stable models, and applying statistical inference to computing weighted stable models. We also present formal comparisons with related formalisms, such as answer set programs, Markov Logic, ProbLog, and P-log.

1 Introduction

Logic programs under the stable model semantics (Gelfond and Lifschitz 1988) is the language of Answer Set Programming (ASP). Many extensions of the stable model semantics have been proposed to incorporate various constructs in knowledge representation. Some of them are related to overcoming the "crisp" or deterministic nature of the stable model semantics by ranking stable models using weak constraints (Buccafurri, Leone, and Rullo 2000), by resolving inconsistencies using Consistency Restoring rules (Balduccini and Gelfond 2003) or possibilistic measure (Bauters et al. 2010), and by assigning probability to stable models (Baral, Gelfond, and Rushton 2009; Nickles and Mileo 2014).

In this paper, we present an alternative approach by introducing the notion of weights into the stable model semantics following the log-linear models of Markov Logic (Richardson and Domingos 2006), a successful approach to combining first-order logic and probabilistic graphical models. Instead of the concept of classical models adopted in Markov Logic, language LP$^{\rm MLN}$ adopts stable models as the logical component. The relationship between LP$^{\rm MLN}$ and Markov Logic is analogous to the known relationship between ASP and SAT. Indeed, many technical results about the relationship between SAT and ASP naturally carry over between LP$^{\rm MLN}$ and Markov Logic. In particular, an implementation of Markov Logic can be used to compute "tight" LP$^{\rm MLN}$ programs, similar to the way "tight" ASP programs can be computed by SAT solvers.

It is also interesting that the relationship between Markov Logic and SAT is analogous to the relationship between LP$^{\rm MLN}$ and ASP: the way that Markov Logic extends SAT in a probabilistic way is similar to the way that LP$^{\rm MLN}$ extends ASP in a probabilistic way. This can be summarized as in the following figure. (The parallel edges imply that the ways that the extensions are defined are similar to each other.)

$$\begin{tikzcd}
\boxed{\text{ASP}} \arrow[r, "/" marking] & \boxed{\text{LP}^{\rm MLN}} \\
\boxed{\text{SAT}} \arrow[u, "//" marking] \arrow[r, "/" marking] & \boxed{\text{MLN}} \arrow[u, "//" marking]
\end{tikzcd}$$

Weighted rules of LP$^{\rm MLN}$ provides a way to resolve inconsistencies among ASP knowledge bases, possibly obtained from different sources with different certainty levels. For example, consider the simple ASP knowledge base KB_1:

$$\begin{array}{rcl}
\mathit{Bird}(x) & \leftarrow & \mathit{ResidentBird}(x) \\
\mathit{Bird}(x) & \leftarrow & \mathit{MigratoryBird}(x) \\
 & \leftarrow & \mathit{ResidentBird}(x), \mathit{MigratoryBird}(x).
\end{array}$$

One data source KB_2 (possibly acquired by some information extraction module) says that *Jo* is a *ResidentBird*:

$$\mathit{ResidentBird}(\mathit{Jo})$$

while another data source KB_3 states that *Jo* is a *MigratoryBird*:

$$\mathit{MigratoryBird}(\mathit{Jo}).$$

The data about *Jo* is actually inconsistent w.r.t. KB_1, so under the (deterministic) stable model semantics, the combined knowledge base $KB = KB_1 \cup KB_2 \cup KB_3$ is not so meaningful. On the other hand, it is still intuitive to conclude that *Jo* is likely a *Bird*, and may be a *ResidentBird* or a *MigratoryBird*. Such reasoning is supported in LP$^{\rm MLN}$.

Under some reasonable assumption, normalized weights of stable models can be understood as probabilities of the stable models. We show that ProbLog (De Raedt, Kimmig, and Toivonen 2007; Fierens et al. 2015) can be viewed as a special case of LP$^{\rm MLN}$. Furthermore, we present a subset of LP$^{\rm MLN}$ where probability is naturally expressed and show how it captures a meaningful fragment of P-log (Baral, Gelfond, and Rushton 2009). In combination of the result that relates $\mathrm{LP}^{\mathrm{MLN}}$ to Markov Logic, the translation from P-log to $\mathrm{LP}^{\mathrm{MLN}}$ yields an alternative, more scalable method for computing the fragment of P-log using standard implementations of Markov Logic.

The paper is organized as follows. After reviewing the deterministic stable model semantics, we define the language $\mathrm{LP}^{\mathrm{MLN}}$ and demonstrate how it can be used for resolving inconsistencies. Then we relate $\mathrm{LP}^{\mathrm{MLN}}$ to each of ASP, Markov Logic, and ProbLog, and define a fragment of $\mathrm{LP}^{\mathrm{MLN}}$ language that allows probability to be represented in a more natural way. Next we show how a fragment of P-log can be turned into that fragment of $\mathrm{LP}^{\mathrm{MLN}}$, and demonstrate the effectiveness of the translation-based computation of the P-log fragment over the existing implementation of P-log.

This paper is an extended version of (Lee and Wang 2015; Lee, Meng, and Wang 2015). The proofs are available from the longer version at `http://reasoning.eas.asu.edu/papers/lpmln-kr-long.pdf`.

2 Review: Stable Model Semantics

We assume a first-order signature σ that contains no function constants of positive arity, which yields finitely many Herbrand interpretations.

We say that a formula is *negative* if every occurrence of every atom in this formula is in the scope of negation.

A *rule* is of the form

$$A \leftarrow B \wedge N \qquad (1)$$

where A is a disjunction of atoms, B is a conjunction of atoms, and N is a negative formula constructed from atoms using conjunction, disjunction and negation. We identify rule (1) with formula $B \wedge N \rightarrow A$. We often use comma for conjunction, semi-colon for disjunction, *not* for negation, as widely used in the literature on logic programming. For example, N could be

$$\neg B_{m+1} \wedge \ldots \wedge \neg B_n \wedge \neg\neg B_{n+1} \wedge \ldots \wedge \neg\neg B_p,$$

which can be also written as

$$\mathit{not}\ B_{m+1}, \ldots, \mathit{not}\ B_n, \mathit{not\ not}\ B_{n+1}, \ldots, \mathit{not\ not}\ B_p.$$

We write $\{A_1\}^{\mathrm{ch}} \leftarrow \mathit{Body}$, where A_1 is an atom, to denote the rule $A_1 \leftarrow \mathit{Body} \wedge \neg\neg A_1$. This expression is called a "choice rule" in ASP. If the head of a rule (A in (1)) is $\bot$, we often omit it and call such a rule *constraint*.

A *logic program* is a finite conjunction of rules. A logic program is called *ground* if it contains no variables.

We say that an Herbrand interpretation I is a *model* of a ground program Π if I satisfies all implications (1) in Π (as in classical logic). Such models can be divided into two groups: "stable" and "non-stable" models, which are distinguished as follows. The *reduct* of Π relative to I, denoted Π^I, consists of "$A \leftarrow B$" for all rules (1) in Π such that $I \models N$. The Herbrand interpretation I is called a *(deterministic) stable model* of Π if I is a minimal Herbrand model of Π^I. (Minimality is understood in terms of set inclusion. We identify an Herbrand interpretation with the set of atoms that are true in it.)

The definition is extended to any non-ground program Π by identifying it with $gr_\sigma[\Pi]$, the ground program obtained from Π by replacing every variable with every ground term of σ.

3 Language $\mathrm{LP}^{\mathrm{MLN}}$

Syntax of $\mathrm{LP}^{\mathrm{MLN}}$

The syntax of $\mathrm{LP}^{\mathrm{MLN}}$ defines a set of weighted rules. More precisely, an $\mathrm{LP}^{\mathrm{MLN}}$ program Π is a finite set of weighted rules $w : R$, where R is a rule of the form (1) and w is either a real number or the symbol α denoting the "infinite weight." We call rule $w : R$ *soft* rule if w is a real number, and *hard* rule if w is α.

We say that an $\mathrm{LP}^{\mathrm{MLN}}$ program is *ground* if its rules contain no variables. We identify any $\mathrm{LP}^{\mathrm{MLN}}$ program Π of signature σ with a ground $\mathrm{LP}^{\mathrm{MLN}}$ program $gr_\sigma[\Pi]$, whose rules are obtained from the rules of Π by replacing every variable with every ground term of σ. The weight of a ground rule in $gr_\sigma[\Pi]$ is the same as the weight of the rule in Π from which the ground rule is obtained. By $\overline{\Pi}$ we denote the unweighted logic program obtained from Π, i.e., $\overline{\Pi} = \{R \mid w : R \in \Pi\}$.

Semantics of $\mathrm{LP}^{\mathrm{MLN}}$

A model of a Markov Logic Network (MLN) does not have to satisfy all formulas in the MLN. For each model, there is a unique maximal subset of the formulas that are satisfied by the model, and the weights of the formulas in that subset determine the probability of the model.

Likewise, a stable model of an $\mathrm{LP}^{\mathrm{MLN}}$ program does not have to be obtained from the whole program. Instead, each stable model is obtained from some subset of the program, and the weights of the rules in that subset determine the probability of the stable model. Unlike MLNs, it may not seem obvious if there is a *unique* maximal subset that derives such a stable model. The following proposition tells us that this is indeed the case, and furthermore that the subset is exactly the set of all rules that are satisfied by I.

Proposition 1 *For any (unweighted) logic program Π and any subset Π' of Π, if I is a stable model of Π' and I satisfies Π, then I is a stable model of Π as well.*

The proposition tells us that if I is a stable model of a program, adding more rules to this program does not affect that I is a stable model of the resulting program as long as I satisfies the rules added. On the other hand, it is clear that I is no longer a stable model if I does not satisfy at least one of the rules added.

For any $\mathrm{LP}^{\mathrm{MLN}}$ program Π, by Π_I we denote the set of rules $w : R$ in Π such that $I \models R$, and by $\mathrm{SM}[\Pi]$ we denote the set $\{I \mid I \text{ is a stable model of } \overline{\Pi_I}\}$. We define the *unnormalized weight* of an interpretation I under Π, denoted $W_\Pi(I)$, as

$$W_\Pi(I) = \begin{cases} exp\left(\displaystyle\sum_{w:R\,\in\,\Pi_I} w\right) & \text{if } I \in \mathrm{SM}[\Pi]; \\ 0 & \text{otherwise.} \end{cases}$$

Notice that $\mathrm{SM}[\Pi]$ is never empty because it always contains $\emptyset$. It is easy to check that $\emptyset$ always satisfies $\overline{\Pi_\emptyset}$, and it is the smallest set that satisfies the reduct $(\overline{\Pi_\emptyset})^\emptyset$.

The *normalized weight* of an interpretation I under Π, denoted $P_\Pi(I)$, is defined as

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_{J\in \mathrm{SM}[\Pi]} W_\Pi(J)}.$$

It is easy to check that normalized weights satisfy the Kolmogorov axioms of probability. So we also call them *probabilities*.

We omit the subscript Π if the context is clear. We say that I is a *(probabilistic) stable model* of Π if $P_\Pi(I) \neq 0$.

The intuition here is similar to that of Markov Logic. For each interpretation I, we try to find a maximal subset (possibly empty) of $\overline{\Pi}$ for which I is a stable model (under the standard stable model semantics). In other words, the LP$^{\mathrm{MLN}}$ semantics is similar to the MLN semantics except that the possible worlds are the *stable* models of some maximal subset of $\overline{\Pi}$, and the probability distribution is over these stable models. Intuitively, $P_\Pi(I)$ indicates how likely to draw I as a stable model of some maximal subset of $\overline{\Pi}$.

For any proposition A, $P_\Pi(A)$ is defined as

$$P_\Pi(A) = \sum_{I:\ I\models A} P_\Pi(I).$$

Conditional probability under Π is defined as usual. For propositions A and B,

$$P_\Pi(A \mid B) = \frac{P_\Pi(A\wedge B)}{P_\Pi(B)}.$$

Often we are interested in stable models that satisfy all hard rules (hard rules encode definite knowledge), in which case the probabilities of stable models can be computed from the weights of the soft rules only, as described below.

For any LP$^{\mathrm{MLN}}$ program Π, by Π^{soft} we denote the set of all soft rules in Π, and by Π^{hard} the set of all hard rules in Π. Let $\mathrm{SM}'[\Pi]$ be the set

$$\{I \mid I \text{ is a stable model of } \overline{\Pi_I} \text{ that satisfy } \overline{\Pi^{\mathrm{hard}}}\ \},$$

and let

$$W'_\Pi(I) = \begin{cases} exp\left(\displaystyle\sum_{w:R\ \in\ (\Pi^{\mathrm{soft}})_I} w\right) & \text{if } I \in \mathrm{SM}'[\Pi]; \\ 0 & \text{otherwise,} \end{cases}$$

$$P'_\Pi(I) = \frac{W'_\Pi(I)}{\sum_{J\in \mathrm{SM}'[\Pi]} W'_\Pi(J)}.$$

Notice the absence of $\lim\limits_{\alpha\to\infty}$ in the definition of $P'_\Pi[I]$. Also, unlike $P_\Pi(I)$, $\mathrm{SM}'[\Pi]$ may be empty, in which case $P'_\Pi(I)$ is not defined. Otherwise, the following proposition tells us that the probability of an interpretation can be computed by considering the weights of the soft rules only.

Proposition 2 *If* $\mathrm{SM}'[\Pi]$ *is not empty, for every interpretation* I*,* $P'_\Pi(I)$ *coincides with* $P_\Pi(I)$*.*

It follows from this proposition that if $\mathrm{SM}'[\Pi]$ is not empty, then every stable model of Π (with non-zero probability) should satisfy all hard rules in Π.

Examples

The weight scheme of LP$^{\mathrm{MLN}}$ provides a simple but effective way to resolve certain inconsistencies in ASP programs.

Example 1 *The example in the introduction can be represented in* LP$^{\mathrm{MLN}}$ *as*

KB_1	$\alpha:$	$\mathit{Bird}(x) \leftarrow \mathit{ResidentBird}(x)$	$(r1)$
	$\alpha:$	$\mathit{Bird}(x) \leftarrow \mathit{MigratoryBird}(x)$	$(r2)$
	$\alpha:$	$\leftarrow \mathit{ResidentBird}(x), \mathit{MigratoryBird}(x)$	$(r3)$
KB_2	$\alpha:$	$\mathit{ResidentBird}(\mathit{Jo})$	$(r4)$
KB_3	$\alpha:$	$\mathit{MigratoryBird}(\mathit{Jo})$	$(r5)$

Assuming that the Herbrand universe is $\{Jo\}$*, the following table shows the weight and the probability of each interpretation.*

I	Π_I	$W_\Pi(I)$	$P_\Pi(I)$
$\emptyset$	$\{r_1, r_2, r_3\}$	$e^{3\alpha}$	0
$\{R(Jo)\}$	$\{r_2, r_3, r_4\}$	$e^{3\alpha}$	0
$\{M(Jo)\}$	$\{r_1, r_3, r_5\}$	$e^{3\alpha}$	0
$\{B(Jo)\}$	$\{r_1, r_2, r_3\}$	0	0
$\{R(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_4\}$	$e^{4\alpha}$	1/3
$\{M(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_5\}$	$e^{4\alpha}$	1/3
$\{R(Jo), M(Jo)\}$	$\{r_4, r_5\}$	$e^{2\alpha}$	0
$\{R(Jo), M(Jo), B(Jo)\}$	$\{r_1, r_2, r_4, r_5\}$	$e^{4\alpha}$	1/3

(The weight of $I = \{\mathit{Bird}(\mathit{Jo})\}$ *is 0 because* I *is not a stable model of* $\overline{\Pi_I}$*.) Thus we can check that*

- $P(\mathit{Bird}(\mathit{Jo})) = 1/3 + 1/3 + 1/3 = 1.$
- $P(\mathit{Bird}(\mathit{Jo}) \mid \mathit{ResidentBird}(\mathit{Jo})) = 1.$
- $P(\mathit{ResidentBird}(\mathit{Jo}) \mid \mathit{Bird}(\mathit{Jo})) = 2/3.$

Instead of α*, one can assign different certainty levels to the additional knowledge bases, such as*

KB'_2	$2:$	$\mathit{ResidentBird}(\mathit{Jo})$	$(r4')$
KB'_3	$1:$	$\mathit{MigratoryBird}(\mathit{Jo})$	$(r5')$

Then the table changes as follows.

I	Π_I	$W_\Pi(I)$	$P_\Pi(I)$
$\emptyset$	$\{r_1, r_2, r_3\}$	$e^{3\alpha}$	$\frac{e^0}{e^2+e^1+e^0}$
$\{R(Jo)\}$	$\{r_2, r_3, r'_4\}$	$e^{2\alpha+2}$	0
$\{M(Jo)\}$	$\{r_1, r_3, r'_5\}$	$e^{2\alpha+1}$	0
$\{B(Jo)\}$	$\{r_1, r_2, r_3\}$	0	0
$\{R(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r'_4\}$	$e^{3\alpha+2}$	$\frac{e^2}{e^2+e^1+e^0}$
$\{M(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r'_5\}$	$e^{3\alpha+1}$	$\frac{e^1}{e^2+e^1+e^0}$
$\{R(Jo), M(Jo)\}$	$\{r'_4, r'_5\}$	e^3	0
$\{R(Jo), M(Jo), B(Jo)\}$	$\{r_1, r_2, r'_4, r'_5\}$	$e^{2\alpha+3}$	0

$P(\mathit{Bird}(\mathit{Jo})) = (e^2+e^1)/(e^2+e^1+e^0) = 0.67 + 0.24$*, so it becomes less certain, though it is still a high chance that we can conclude that Jo is a Bird.*

Notice that the weight changes not only affect the probability, but also the stable models (having non-zero probabilities) themselves: Instead of $\{R(Jo), M(Jo), B(Jo)\}$*, the empty set is a stable model of the new program.*

Assigning a different certainty level to each rule affects the probability associated with each stable model, representing how certain we can derive the stable model from the knowledge base. This could be useful as more incoming data reinforces the certainty levels of the information.

Remark. In some sense, the distinction between soft rules and hard rules in $\mathrm{LP}^{\mathrm{MLN}}$ is similar to the distinction CR-Prolog (Balduccini and Gelfond 2003) makes between consistency-restoring rules (CR-rules) and standard ASP rules: some CR-rules are added to the standard ASP program part until the resulting program has a stable model. On the other hand, CR-Prolog has little to say when the ASP program has no stable models no matter what CR-rules are added (**c.f.** Example 1).

Example 2 *"Markov Logic has the drawback that it cannot express (non-ground) inductive definitions" (Fierens et al. 2015) because it relies on classical models. This is not the case with* $\mathrm{LP}^{\mathrm{MLN}}$. *For instance, consider that x may influence y if x is a friend to y, and the influence relation is a minimal relation that is closed under transitivity.*

$$\begin{array}{l}\alpha : \mathit{Friend}(A,B)\\ \alpha : \mathit{Friend}(B,C)\\ 1 : \mathit{Influence}(x,y) \leftarrow \mathit{Friend}(x,y)\\ \alpha : \mathit{Influence}(x,y) \leftarrow \mathit{Influence}(x,z), \mathit{Influence}(z,y).\end{array}$$

Note that the third rule is soft: a person does not necessarily influence his/her friend. The fourth rule says if x influences z, and z influences y, we can say x influences y. On the other hand, we do not want this relation to be vacuously true.

Assuming that there are only three people A, B, C in the domain (thus there are $1+1+9+27$ ground rules), there are four stable models with non-zero probabilities. Let $Z = e^9 + 2e^8 + e^7$. (Fr abbreviates for Friend and Inf for Influence)

- $I_1 = \{\mathit{Fr}(A,B), \mathit{Fr}(B,C), \mathit{Inf}(A,B), \mathit{Inf}(B,C), \mathit{Inf}(A,C)\}$ *with probability* e^9/Z.
- $I_2 = \{\mathit{Fr}(A,B), \mathit{Fr}(B,C), \mathit{Inf}(A,B)\}$ *with probability* e^8/Z.
- $I_3 = \{\mathit{Fr}(A,B), \mathit{Fr}(B,C), \mathit{Inf}(B,C)\}$ *with probability* e^8/Z.
- $I_4 = \{\mathit{Fr}(A,B), \mathit{Fr}(B,C)\}$ *with probability* e^7/Z.

Thus we get

- $P(\mathit{Inf}(A,B)) = P(\mathit{Inf}(B,C)) = (e^9 + e^8)/Z = 0.7311.$
- $P(\mathit{Inf}(A,C)) = e^9/Z = 0.5344.$

Increasing the weight of the third rule yields higher probabilities for deriving $\mathit{Influence}(A,B)$, $\mathit{Influence}(B,C)$, and $\mathit{Influence}(A,C)$. Still, the first two have the same probability, and the third has less probability than the first two.

4 Relating $\mathrm{LP}^{\mathrm{MLN}}$ to ASP

Any logic program under the stable model semantics can be turned into an $\mathrm{LP}^{\mathrm{MLN}}$ program by assigning the infinite weight to every rule. That is, for any logic program $\Pi = \{R_1, \dots, R_n\}$, the corresponding $\mathrm{LP}^{\mathrm{MLN}}$ program $\mathbb{P}_\Pi$ is $\{\alpha : R_1, \dots, \alpha : R_n\}$.

Theorem 1 *For any logic program Π, the (deterministic) stable models of Π are exactly the (probabilistic) stable models of $\mathbb{P}_\Pi$ whose weight is $e^{k\alpha}$, where k is the number of all (ground) rules in Π. If Π has at least one stable model, then all stable models of $\mathbb{P}_\Pi$ have the same probability, and are thus the stable models of Π as well.*

Weak Constraints and $\mathrm{LP}^{\mathrm{MLN}}$

The idea of softening rules in $\mathrm{LP}^{\mathrm{MLN}}$ is similar to the idea of *weak constraints* in ASP, which is used for certain optimization problems. A weak constraint has the form " $:\sim$ *Body* [*Weight* : *Level*]." The stable models of a program Π (whose rules have the form (1)) plus a set of weak constraints are the stable models of Π with the minimum penalty, where a penalty is calculated from *Weight* and *Level* of violated weak constraints.

Since levels can be compiled into weights (Buccafurri, Leone, and Rullo 2000), we consider weak constraints of the form

$$:\sim \mathit{Body}\ \ [\mathit{Weight}] \tag{2}$$

where *Weight* is a positive integer. We assume all weak constraints are grounded. The penalty of a stable model is defined as the sum of the weights of all weak constraints whose bodies are satisfied by the stable model.

Such a program can be turned into an $\mathrm{LP}^{\mathrm{MLN}}$ program as follows. Each weak constraint (2) is turned into

$$-w : \ \bot \leftarrow \neg \mathit{Body}.$$

The standard ASP rules are identified with hard rules in $\mathrm{LP}^{\mathrm{MLN}}$. For example, the program with weak constraints

$$\begin{array}{ll} a \vee b & :\sim a\ \ [1]\\ c \leftarrow b & :\sim b\ \ [1]\\ & :\sim c\ \ [1]\end{array}$$

is turned into

$$\begin{array}{llll} \alpha : & a \vee b & -1 : & \bot \leftarrow \neg a\\ \alpha : & c \leftarrow b & -1 : & \bot \leftarrow \neg b\\ & & -1 : & \bot \leftarrow \neg c.\end{array}$$

The $\mathrm{LP}^{\mathrm{MLN}}$ program has two stable models: $\{a\}$ with the normalized weight $\frac{e^{-1}}{e^{-1}+e^{-2}}$ and $\{b, c\}$ with the normalized weight $\frac{e^{-2}}{e^{-1}+e^{-2}}$. The former, with the larger normalized weight, is the stable model of the original program containing the weak constraints.

Proposition 3 *For any program with weak constraints that has a stable model, its stable models are the same as the stable models of the corresponding* $\mathrm{LP}^{\mathrm{MLN}}$ *program with the highest normalized weight.*

5 Relating $\mathrm{LP}^{\mathrm{MLN}}$ to MLNs

Embedding MLNs in $\mathrm{LP}^{\mathrm{MLN}}$

Similar to the way that SAT can be embedded in ASP, Markov Logic can be easily embedded in $\mathrm{LP}^{\mathrm{MLN}}$. More precisely, any MLN $\mathbb{L}$ can be turned into an $\mathrm{LP}^{\mathrm{MLN}}$ program $\Pi_\mathbb{L}$ so that the models of $\mathbb{L}$ coincide with the stable models of $\Pi_\mathbb{L}$ while retaining the same probability distribution.

$\mathrm{LP}^{\mathrm{MLN}}$ program $\Pi_\mathbb{L}$ is obtained from $\mathbb{L}$ by turning each weighted formula $w : F$ into weighted rule $w : \ \bot \leftarrow \neg F$ and adding

$$w : \ \{A\}^{\mathrm{ch}}$$

for every ground atom A of σ and any weight w. The effect of adding the choice rules is to exempt A from minimization under the stable model semantics.

Theorem 2 *Any MLN $\mathbb{L}$ and its* $\mathrm{LP}^{\mathrm{MLN}}$ *representation* $\Pi_{\mathbb{L}}$ *have the same probability distribution over all interpretations.*

The embedding tells us that the exact inference in $\mathrm{LP}^{\mathrm{MLN}}$ is at least as hard as the one in MLNs, which is #P-hard. In fact, it is easy to see that when all rules in $\mathrm{LP}^{\mathrm{MLN}}$ are non-disjunctive, counting the stable models of $\mathrm{LP}^{\mathrm{MLN}}$ is in #P, which yields that the exact inference for non-disjunctive $\mathrm{LP}^{\mathrm{MLN}}$ programs is #P-complete. Therefore, approximation algorithms, such as Gibbs sampling, may be desirable for computing large $\mathrm{LP}^{\mathrm{MLN}}$ programs. The next section tells us that we can apply the MLN approximation algorithms to computing $\mathrm{LP}^{\mathrm{MLN}}$ based on the reduction of the latter to the former.

Completion: Turning $\mathrm{LP}^{\mathrm{MLN}}$ to MLN

It is known that the stable models of a tight logic program coincide with the models of the program's completion (Erdem and Lifschitz 2003). This yielded a way to compute stable models using SAT solvers. The method can be extended to $\mathrm{LP}^{\mathrm{MLN}}$ so that probability queries involving the stable models can be computed using existing implementations of MLNs, such as Alchemy (`http://alchemy.cs.washington.edu`).

We define the *completion* of Π, denoted $\mathit{Comp}(\Pi)$, to be the MLN which is the union of Π and the hard formula

$$\alpha:\ A \rightarrow \bigvee_{\substack{w:A_1\vee\cdots\vee A_k\leftarrow Body\in\ \Pi \\ A\in\{A_1,\dots,A_k\}}} \Big(Body \wedge \bigwedge_{A'\in\{A_1,\dots,A_k\}\setminus\{A\}} \neg A'\Big)$$

for each ground atom A.

This is a straightforward extension of the completion from (Lee and Lifschitz 2003) by simply assigning the infinite weight α to the completion formulas. Likewise, we say that $\mathrm{LP}^{\mathrm{MLN}}$ program Π is *tight* if $\overline{\Pi}$ is tight according to (Lee and Lifschitz 2003), i.e., the positive dependency graph of $\overline{\Pi}$ is acyclic.

Theorem 3 *For any tight* $\mathrm{LP}^{\mathrm{MLN}}$ *program* Π *such that* $\mathrm{SM}'[\Pi]$ *is not empty,* Π *(under the* $\mathrm{LP}^{\mathrm{MLN}}$ *semantics) and* $\mathit{Comp}(\Pi)$ *(under the MLN semantics) have the same probability distribution over all interpretations.*

The theorem can be generalized to non-tight programs by considering loop formulas (Lin and Zhao 2004), which we skip here for brevity.

6 Relation to ProbLog

It turns out that $\mathrm{LP}^{\mathrm{MLN}}$ is a proper generalization of ProbLog, a well-developed probabilistic logic programming language that is based on the distribution semantics by Sato (1995).

Review: ProbLog

The review follows (Fierens et al. 2015). As before, we identify a non-ground ProbLog program with its ground instance. So for simplicity we restrict attention to ground ProbLog programs only.

In ProbLog, ground atoms over σ are divided into two groups: *probabilistic* atoms and *derived* atoms. A *(ground) ProbLog program* $\mathbb{P}$ is a tuple $\langle \mathit{PF}, \Pi\rangle$, where

- *PF* is a set of ground probabilistic facts of the form $pr :: a$,
- Π is a set of ground rules of the following form
  $$A \leftarrow B_1,\dots,B_m, \mathit{not}\ B_{m+1},\dots,\mathit{not}\ B_n$$
  where A, $B_1,\dots B_n$ are atoms from σ ($0 \le m \le n$), and A is not a probabilistic atom.

Probabilistic atoms act as random variables and are assumed to be independent from each other. A *total choice TC* is any subset of the probabilistic atoms. Given a total choice $\mathit{TC} = \{a_1,\dots,a_m\}$, the *probability* of a total choice *TC*, denoted $\mathit{Pr}_{\mathbb{P}}(\mathit{TC})$, is defined as

$$pr(a_1)\times\ldots\times pr(a_m)\times(1-pr(b_1))\times\ldots\times(1-pr(b_n))$$

where $b_1,\dots,b_n$ are probabilistic atoms not belonging to *TC*, and each of $pr(a_i)$ and $pr(b_j)$ is the probability assigned to a_i and b_j according to the set *PF* of ground probabilistic atoms.

The ProbLog semantics is only well-defined for programs $\mathbb{P} = \langle \mathit{PF}, \Pi\rangle$ such that $\Pi \cup \mathit{TC}$ has a "total" (two-valued) well-founded model for each total choice *TC*. Given such $\mathbb{P}$, the probability of an interpretation I, denoted $P_{\mathbb{P}}(I)$, is defined as $\mathit{Pr}_{\mathbb{P}}(\mathit{TC})$ if there exists a total choice *TC* such that I is the total well-founded model of $\Pi\cup\mathit{TC}$, and 0 otherwise.

ProbLog as a Special Case of $\mathrm{LP}^{\mathrm{MLN}}$

Given a ProbLog program $\mathbb{P} = \langle \mathit{PF}, \Pi\rangle$, we construct the corresponding $\mathrm{LP}^{\mathrm{MLN}}$ program $\mathbb{P}'$ as follows:

- For each probabilistic fact $pr :: a$ in $\mathbb{P}$, $\mathrm{LP}^{\mathrm{MLN}}$ program $\mathbb{P}'$ contains (i) $\mathit{ln}(pr) : a$ and $\mathit{ln}(1-pr) :\ \leftarrow a$ if $0 < pr < 1$; (ii) $\alpha : a$ if $pr = 1$; (iii) $\alpha :\ \leftarrow a$ if $pr = 0$.
- For each rule $R \in \Pi$, $\mathbb{P}'$ contains $\alpha : R$. In other words, R is identified with a hard rule in $\mathbb{P}'$.

Theorem 4 *Any well-defined ProbLog program* $\mathbb{P}$ *and its* $\mathrm{LP}^{\mathrm{MLN}}$ *representation* $\mathbb{P}'$ *have the same probability distribution over all interpretations.*

Example 3 *Consider the ProbLog program*

$$\begin{array}{lcl} 0.6 :: p & \qquad\qquad & r \leftarrow p \\ 0.3 :: q & & r \leftarrow q \end{array}$$

which can be identified with the $\mathrm{LP}^{\mathrm{MLN}}$ *program*

$$\begin{array}{lll} \mathit{ln}(0.6) :\ p & \mathit{ln}(0.3) :\ q & \alpha :\ r \leftarrow p \\ \mathit{ln}(0.4) :\ \leftarrow p & \mathit{ln}(0.7) :\ \leftarrow q & \alpha :\ r \leftarrow q \end{array}$$

Syntactically, $\mathrm{LP}^{\mathrm{MLN}}$ allows more general rules than ProbLog, such as disjunctions in the head, as well as the empty head and double negations in the body. Further, $\mathrm{LP}^{\mathrm{MLN}}$ allows rules to be weighted as well as facts, and do not distinguish between probabilistic facts and derived atoms. Semantically, while the ProbLog semantics is based on well-founded models, $\mathrm{LP}^{\mathrm{MLN}}$ handles stable model reasoning for more general classes of programs. Unlike ProbLog which is only well-defined when each total choice leads to a unique well-founded model, $\mathrm{LP}^{\mathrm{MLN}}$ can handle multiple stable models in a flexible way similar to the way MLN handles multiple models.

7 Multi-Valued Probabilistic Programs

In this section we define a simple fragment of LP$^{\rm MLN}$ that allows us to represent probability in a more natural way. For simplicity of the presentation, we will assume a propositional signature. An extension to first-order signatures is straightforward.

We assume that the propositional signature σ is constructed from "constants" and their "values." A *constant* c is a symbol that is associated with a finite set $Dom(c)$, called the *domain*. The signature σ is constructed from a finite set of constants, consisting of atoms $c\!=\!v$ [1] for every constant c and every element v in $Dom(c)$. If the domain of c is $\{\mathbf{f},\mathbf{t}\}$ then we say that c is *Boolean*, and abbreviate $c\!=\!\mathbf{t}$ as c and $c\!=\!\mathbf{f}$ as $\sim\!c$.

We assume that constants are divided into *probabilistic* constants and *regular* constants. A multi-valued probabilistic program $\mathbf{\Pi}$ is a tuple $\langle PF, \Pi\rangle$, where

- PF contains *probabilistic constant declarations* of the following form:
$$p_1 : c\!=\!v_1 \mid \cdots \mid p_n : c\!=\!v_n \qquad (3)$$
one for each probabilistic constant c, where $\{v_1,\dots,v_n\} = Dom(c)$, $v_i \neq v_j$, $0 \le p_1,\dots,p_n \le 1$ and $\sum_{i=1}^n p_i = 1$. We use $M_{\mathbf{\Pi}}(c = v_i)$ to denote p_i. In other words, PF describes the probability distribution over each "random variable" c.
- Π is a set of rules of the form (1) such that A contains no probabilistic constants.

The semantics of such a program $\mathbf{\Pi}$ is defined as a shorthand for LP$^{\rm MLN}$ program $T(\mathbf{\Pi})$ of the same signature as follows.

- For each probabilistic constant declaration (3), $T(\mathbf{\Pi})$ contains, for each $i = 1,\dots,n$, (i) $ln(p_i)\ :\ c = v_i$ if $0 < p_i < 1$; (ii) $\alpha :\ c\!=\!v_i$ if $p_i = 1$; (iii) $\alpha : \leftarrow c\!=\!v_i$ if $p_i = 0$.
- For each rule in Π of form (1), $T(\mathbf{\Pi})$ contains
$$\alpha :\ \ A \leftarrow B, N.$$
- For each constant c, $T(\mathbf{\Pi})$ contains the uniqueness of value constraints
$$\alpha :\ \ \bot \leftarrow c\!=\!v_1 \wedge c = v_2 \qquad (4)$$
for all $v_1, v_2 \in Dom(c)$ such that $v_1 \neq v_2$. For each probabilistic constant c, $T(\mathbf{\Pi})$ also contains the existence of value constraint
$$\alpha :\ \ \bot \leftarrow \neg \bigvee_{v\in Dom(c)} c\!=\!v\ . \qquad (5)$$
This means that a regular constant may be undefined (i.e., have no values associated with it), while a probabilistic constant is always associated with some value.

Example 4 *The multi-valued probabilistic program*

$$\begin{array}{l} 0.25 : Outcome\!=\!6 \mid 0.15 : Outcome\!=\!5 \\ \quad \mid 0.15 : Outcome\!=\!4 \mid 0.15 : Outcome\!=\!3 \\ \quad\ \mid 0.15 : Outcome\!=\!2 \mid 0.15 : Outcome\!=\!1 \\ Win \leftarrow Outcome\!=\!6. \end{array}$$

is understood as shorthand for the LP$^{\rm MLN}$ *program*

$$\begin{array}{rll} ln(0.25): & Outcome\!=\!6 & \\ ln(0.15): & Outcome\!=\!i & (i = 1,\dots,5) \\ \alpha: & Win \leftarrow Outcome\!=\!6 & \\ \alpha: & \bot \leftarrow Outcome\!=\!i \wedge Outcome\!=\!j & (i \neq j) \\ \alpha: & \bot \leftarrow \neg \bigvee_{i=1,\dots 6} Outcome\!=\!i. & \end{array}$$

We say an interpretation of $\mathbf{\Pi}$ is *consistent* if it satisfies the hard rules (4) for every constant and (5) for every probabilistic constant. For any consistent interpretation I, we define the set $TC(I)$ ("Total Choice") to be $\{c = v \mid c \text{ is a probabilistic constant such that } c = v \in I\}$ and define

$$\begin{array}{l} \text{SM}''[\mathbf{\Pi}] = \{I \mid I \text{ is consistent} \\ \qquad\qquad\qquad \text{and is a stable model of } \Pi \cup TC(I)\}. \end{array}$$

For any interpretation I, we define

$$W''_{\mathbf{\Pi}}(I) = \begin{cases} \prod\limits_{c=v\,\in\, TC(I)} M_{\mathbf{\Pi}}(c = v) & \text{if } I \in \text{SM}''[\mathbf{\Pi}] \\ 0 & \text{otherwise} \end{cases}$$

and

$$P''_{\mathbf{\Pi}}(I) = \frac{W''_{\mathbf{\Pi}}(I)}{\sum_{J\in SM''[\mathbf{\Pi}]} W''_{\mathbf{\Pi}}(J)}.$$

The following proposition tells us that the probability of an interpretation can be computed from the probabilities assigned to probabilistic atoms, similar to the way ProbLog is defined.

Proposition 4 *For any multi-valued probabilistic program* $\mathbf{\Pi}$ *such that each* p_i *in* (3) *is positive for every probabilistic constant* c*, if* SM$''[\mathbf{\Pi}]$ *is not empty, then for any interpretation* I*,* $P''_{\mathbf{\Pi}}(I)$ *coincides with* $P_{T(\mathbf{\Pi})}(I)$.

8 P-log and LP$^{\rm MLN}$

Simple P-log

In this section, we define a fragment of P-log, which we call *simple P-log*.

Syntax Let σ be a multi-valued propositional signature as in the previous section. A simple P-log program Π is a tuple

$$\Pi = \langle R, S, P, Obs, Act\rangle \qquad (6)$$

where

- R is a set of normal rules of the form
$$A \leftarrow B_1,\dots,B_m, not\ B_{m+1},\dots, not\ B_n. \qquad (7)$$
Here and after we assume $A, B_1,\dots,B_n$ are atoms from σ ($0 \le m \le n$).
- S is a set of *random selection rules* of the form
$$[r]\ \ random(c) \leftarrow B_1,\dots,B_m, not\ B_{m+1},\dots, not\ B_n \qquad (8)$$
where r is an identifier and c is a constant.
- P is a set of *probability atoms (pr-atoms)* of the form
$$pr_r(c\!=\!v \mid B_1,\dots,B_m, not\ B_{m+1},\dots, not\ B_n) = p$$
where r is the identifier of some random selection rule in S, c is a constant, and $v \in Dom(c)$, and $p \in [0,1]$.

[1]Note that here "=" is just a part of the symbol for propositional atoms, and is not equality in first-order logic.

- Obs is a set of atomic facts of the form $Obs(c\!=\!v)$ where c is a constant and $v \in Dom(c)$.
- Act is a set of atomic facts of the form $Do(c\!=\!v)$ where c is a constant and $v \in Dom(c)$.

Example 5 *We use the following simple P-log program as our main example ($d \in \{D_1, D_2\}$, $y \in \{1, \dots 6\}$):*

$$\begin{array}{c} Owner(D_1)\!=\!Mike \\ Owner(D_2)\!=\!John \\ Even(d) \leftarrow Roll(d)\!=\!y,\ y \bmod 2 = 0 \\ {\sim}Even(d) \leftarrow not\ Even(d) \\ [r(d)]\ random(Roll(d)) \\ pr(Roll(d)\!=\!6 \mid Owner(d)\!=\!Mike) = \frac{1}{4}. \end{array}$$

Semantics Given a simple P-log program Π of the form (6), a (standard) ASP program $\tau(\Pi)$ with the multi-valued signature σ' is constructed as follows:

- σ' contains all atoms in σ, and atom $Intervene(c)\!=\!\mathbf{t}$ (abbreviated as $Intervene(c)$) for every constant c of σ; the domain of $Intervene(c)$ is $\{\mathbf{t}\}$.
- $\tau(\Pi)$ contains all rules in R.
- For each random selection rule of the form (8) with $Dom(c) = \{v_1, \dots, v_n\}$, $\tau(\Pi)$ contains the following rules:

  $$\begin{array}{l} c\!=\!v_1; \dots; c\!=\!v_n \leftarrow \\ \quad B_1, \dots, B_m, not\ B_{m+1}, \dots, not\ B_n, not\ Intervene(c). \end{array}$$
- $\tau(\Pi)$ contains all atomic facts in Obs and Act.
- For every atom $c\!=\!v$ in σ,

  $$\leftarrow Obs(c\!=\!v), not\ c\!=\!v.$$
- For every atom $c\!=\!v$ in σ, $\tau(\Pi)$ contains

  $$\begin{array}{c} c\!=\!v \leftarrow Do(c\!=\!v) \\ Intervene(c) \leftarrow Do(c\!=\!v). \end{array}$$

Example 5 continued *The following is $\tau(\Pi)$ for the simple P-log program Π in Example 5 ($x \in \{Mike, John\}$, $b \in \{\mathbf{t}, \mathbf{f}\}$):*

$$\begin{array}{c} Owner(D_1)\!=\!Mike \\ Owner(D_2)\!=\!John \\ Even(d) \leftarrow Roll(d)\!=\!y,\ y \bmod 2 = 0 \\ {\sim}Even(d) \leftarrow not\ Even(d) \\ Roll(d)\!=\!1; Roll(d)\!=\!2; Roll(d)\!=\!3; Roll(d)\!=\!4; \\ Roll(d)\!=\!5; Roll(d)\!=\!6 \leftarrow not\ Intervene(Roll(d)) \\ \leftarrow Obs(Owner(d)\!=\!x), not\ Owner(d)\!=\!x \\ \leftarrow Obs(Even(d)\!=\!b), not\ Even(d)\!=\!b \\ \leftarrow Obs(Roll(d)\!=\!y), not\ Roll(d)\!=\!y \\ Owner(d)\!=\!x \leftarrow Do(Owner(d)\!=\!x) \\ Even(d)\!=\!b \leftarrow Do(Even(d)\!=\!b) \\ Roll(d)\!=\!y \leftarrow Do(Roll(d)\!=\!y) \\ Intervene(Owner(d)) \leftarrow Do(Owner(d)\!=\!x) \\ Intervene(Even(d)) \leftarrow Do(Even(d)\!=\!b) \\ Intervene(Roll(d)) \leftarrow Do(Roll(d)\!=\!y). \end{array}$$

The stable models of $\tau(\Pi)$ are called the *possible worlds* of Π, and denoted by $\omega(\Pi)$. For an interpretation W and an atom $c\!=\!v$, we say $c\!=\!v$ is *possible* in W with respect to Π if Π contains a random selection rule for c

$$[r]\ \ random(c) \leftarrow B,$$

where B is a set of atoms possibly preceded with *not*, and W satisfies B. We say r is *applied* in W if $W \models B$.

We say that a pr-atom $pr_r(c\!=\!v \mid B) = p$ is *applied* in W if $W \models B$ and r is applied in W.

As in (Baral, Gelfond, and Rushton 2009), we assume that simple P-log programs Π satisfy the following conditions:

- **Unique random selection rule** For any constant c, program Π contains at most one random selection rule for c that is applied in W.
- **Unique probability assignment** If Π contains a random selection rule r for constant c that is applied in W, then, for any two different probability atoms

  $$\begin{array}{c} pr_r(c\!=\!v_1 \mid B') = p_1 \\ pr_r(c\!=\!v_2 \mid B'') = p_2 \end{array}$$

  in Π that are applied in W, we have $v_1 \neq v_2$ and $B' = B''$.

Given a simple P-log program Π, a possible world $W \in \omega(\Pi)$ and a constant c for which $c\!=\!v$ is possible in W, we first define the following notations:

- Since $c = v$ is possible in W, by the unique random selection rule assumption, it follows that there is exactly one random selection rule r for constant c that is applied in W. Let $r_{W,c}$ denote this random selection rule. By the unique probability assignment assumption, if there are pr-atoms of the form $pr_{r_{W,c}}(c\!=\!v \mid B)$ that are applied in W, all B in those pr-atoms should be the same. We denote this B by $B_{W,c}$. Define $PR_W(c)$ as

  $$\{pr_{r_{W,c}}(c\!=\!v \mid B_{W,c}) = p \in \Pi \mid v \in Dom(c)\}.$$

  if $W \not\models Intervene(c)$ and $\emptyset$ otherwise.
- Define $AV_W(c)$ as

  $$\left\{v \mid pr_{r_{W,c}}(c\!=\!v \mid B_{W,c}) = p \in PR_W(c)\right\}.$$
- For each $v \in AV_W(c)$, define the *assigned probability* of $c\!=\!v$ w.r.t. W, denoted by $ap_W(c\!=\!v)$, as the value p for which $pr_{r_{W,c}}(c\!=\!v \mid B_{W,c}) = p \in PR_W(c)$.
- Define the *default probability* for c w.r.t. W, denoted by $dp_W(c)$, as

  $$dp_W(c) = \frac{1 - \sum_{v \in AV_W(c)} ap_W(c\!=\!v)}{|Dom(c) \setminus AV_W(c)|}.$$

For every possible world $W \in \omega(\Pi)$ and every atom $c\!=\!v$ possible in W, the causal probability $P(W, c\!=\!v)$ is defined as follows:

$$P(W, c\!=\!v) = \begin{cases} ap_W(c\!=\!v) & \text{if } v \in AV_W(c) \\ dp_W(c) & \text{otherwise.} \end{cases}$$

The *unnormalized probability* of a possible world W, denoted by $\hat{\mu}_\Pi(W)$, is defined as

$$\hat{\mu}_\Pi(W) = \prod_{\substack{c=v \in W \text{ and} \\ c=v \text{ is possible in } W}} P(W, c\!=\!v).$$

Assuming Π has at least one possible world with nonzero unnormalized probability, the *normalized probability* of W, denoted by $\mu_\Pi(W)$, is defined as

$$\mu_\Pi(W) = \frac{\hat{\mu}_\Pi(W)}{\sum_{W_i \in \omega(\Pi)} \hat{\mu}_\Pi(W_i)}.$$

Given a simple P-log program Π and a formula A, the probability of A with respect to Π is defined as

$$P_\Pi(A) = \sum_{W \text{ is a possible world of } \Pi \text{ that satisfies } A} \mu_\Pi(W).$$

We say Π is *consistent* if Π has at least one possible world.

Example 5 continued *Given the possible world* $W = \{Owner(D_1) = Mike, Owner(D_2) = John, Roll(D_1) = 6, Roll(D_2) = 3, Even(D_1)\}$*, the probability of* $Roll(D_1) = 6$ *is* $P(W, Roll(D_1) = 6) = 0.25$*, the probability of* $Roll(D_2) = 3$ *is* $\frac{1}{6}$*. The unnormalized probability of* W*, i.e.,* $\hat{\mu}(W) = P(W, Roll(D_1) = 6) \cdot P(W, Roll(D_2) = 3) = \frac{1}{24}$*.*

The main differences between simple P-log and P-log are as follows.

- The unique probability assignment assumption in P-log is more general: it does not require the part $B' = B''$. However, all the examples in the P-log paper (Baral, Gelfond, and Rushton 2009) satisfy our stronger unique probability assignment assumption.
- P-log allows a more general random selection rule of the form
$$[r]\; random(c : \{x : P(x)\}) \leftarrow B'.$$
Among the examples in (Baral, Gelfond, and Rushton 2009), only the "Monty Hall Problem" encoding and the "Moving Robot Problem" encoding use "dynamic range $\{x : P(x)\}$" in random selection rules and cannot be represented as simple P-log programs.

Turning Simple P-log into Multi-Valued Probabilistic Programs

The main idea of the syntactic translation is to introduce auxiliary probabilistic constants for encoding the assigned probability and the default probability.

Given a simple P-log program Π, a constant c, a set of literals B,[2] and a random selection rule $[r]\; random(c) \leftarrow B'$ in Π, we first introduce several notations, which resemble the ones used for defining the P-log semantics.

- We define $PR_{B,r}(c)$ as
$$\{pr_r(c{=}v \mid B) = p \in \Pi \mid v \in Dom(c)\}$$
if *Act* in Π does not contain $Do(c = v')$ for any $v' \in Dom(c)$ and $\emptyset$ otherwise.
- We define $AV_{B,r}(c)$ as
$$\{v \mid pr_r(c{=}v \mid B) = p \in PR_{B,r}(c)\}.$$
- For each $v \in AV_{B,r}(c)$, we define the *assigned probability* of $c = v$ w.r.t. B, r, denoted by $ap_{B,r}(c = v)$, as the value p for which $pr_r(c{=}v \mid B) = p \in PR_{B,r}(c)$.

[2]A literal is either an atom A or its negation *not* A.

- We define the *default probability* for c w.r.t. B and r, denoted by $dp_{B,r}(c)$, as
$$dp_{B,r}(c) = \frac{1 - \sum_{v \in AV_{B,r}(c)} ap_{B,r}(c{=}v)}{|Dom(c) \setminus AV_{B,r}(c)|}.$$
- For each $c \in v$, define its *causal probability* w.r.t. B and r, denoted by $P(B, r, c{=}v)$, as
$$P(B, r, c{=}v) = \begin{cases} ap_{B,r}(c{=}v) & \text{if } v \in AV_{B,r}(c) \\ dp_{B,r}(c) & \text{otherwise.} \end{cases}$$

Now we translate Π into the corresponding multi-valued probabilistic program $\Pi^{\text{LP}^{\text{MLN}}}$ as follows:

- The signature of $\Pi^{\text{LP}^{\text{MLN}}}$ is
$$\begin{aligned} &\sigma' \cup \{pf^c_{B,r}{=}v \mid PR_{B,r}(c) \neq \emptyset \text{ and } v \in Dom(c)\} \\ &\quad \cup \{pf^c_{\Box,r}{=}v \mid r \text{ is a random selection rule of } \Pi \text{ for } c \\ &\qquad\qquad \text{and } v \in Dom(c)\} \\ &\quad \cup \{Assigned_r{=}\mathbf{t} \mid r \text{ is a random selection rule of } \Pi\}. \end{aligned}$$
- $\Pi^{\text{LP}^{\text{MLN}}}$ contains all rules in $\tau(\Pi)$.
- For any constant c, any random selection rule r for c, and any set B of literals such that $PR_{B,r}(c) \neq \emptyset$, include in $\Pi^{\text{LP}^{\text{MLN}}}$:
  - the probabilistic constant declaration:
  $$P(B, r, c{=}v_1) : pf^c_{B,r}{=}v_1 \mid \dots \mid P(B, r, c{=}v_n) : pf^c_{B,r}{=}v_n$$
  for each probabilistic constant $pf^c_{B,r}$ of the signature, where $\{v_1, \dots, v_n\} = Dom(c)$. The constant $pf^c_{B,r}$ is used for representing the probability distribution for c when condition B holds in the experiment represented by r.
  - the rules
  $$c{=}v \leftarrow B, B', pf^c_{B,r}{=}v, \textit{not Intervene}(c). \qquad (9)$$
  for all $v \in Dom(c)$, where B' is the body of the random selection rule r. These rules assign v to c when the assigned probability distribution applies to $c{=}v$.
  - the rule
  $$Assigned_r \leftarrow B, B', \textit{not Intervene}(c)$$
  where B' is the body of the random selection rule r (we abbreviate $Assigned_r{=}\mathbf{t}$ as $Assigned_r$). $Assigned_r$ becomes true when any pr-atoms for c related to r is applied.
- For any constant c and any random selection rule r for c, include in $\Pi^{\text{LP}^{\text{MLN}}}$:
  - the probabilistic constant declaration
  $$\frac{1}{|Dom(c)|} : pf^c_{\Box,r}{=}v_1 \mid \cdots \mid \frac{1}{|Dom(c)|} : pf^c_{\Box,r}{=}v_n$$
  for each probabilistic constant $pf^c_{\Box,r}$ of the signature, where $\{v_1, \dots, v_n\} = Dom(c)$. The constant $pf^c_{\Box,r}$ is used for representing the default probability distribution for c when there is no applicable pr-atom.

Example	**Parameter**	**plog1**	**plog2**	**Alchemy** (default)	**Alchemy** (maxstep=5000)
dice	$N_{dice}=2$	$0.00s+0.00s$[a]	$0.00s+0.00s$[b]	$0.02s+0.21s$[c]	$0.02s+0.96s$
	$N_{dice}=7$	$1.93s+31.37s$	$0.00s+1.24s$	$0.13s+0.73s$	$0.12s+3.39s$
	$N_{dice}=8$	$12.66s+223.02s$	$0.00s+6.41s$	$0.16s+0.84s$	$0.16s+3.86s$
	$N_{dice}=9$	timeout	$0.00s+48.62s$	$0.19s+0.95s$	$0.19s+4.37s$
	$N_{dice}=10$	timeout	timeout	$0.23s+1.06s$	$0.24s+4.88s$
	$N_{dice}=100$	timeout	timeout	$19.64s+16.34s$	$19.55s+76.18s$
robot	$maxstep=5$	$0.00s+0.00s$	segment fault	$2.34s+2.54s$	$2.3s+11.75s$
	$maxstep=10$	$0.37s+4.86s$	segment fault	$4.78s+5.24s$	$4.74s+24.34s$
	$maxstep=12$	$3.65+51.76s$	segment fault	$5.72s+6.34s$	$5.75s+29.46s$
	$maxstep=13$	$11.68s+168.15s$	segment fault	$6.2s+6.89s$	$6.2s+31.96s$
	$maxstep=15$	timeout	segment fault	$7.18s+7.99s$	$7.34s+37.67s$
	$maxstep=20$	timeout	segment fault	$9.68s+10.78s$	$9.74s+50.04s$

Table 1: Performance Comparison between Two Ways to Compute Simple P-log Programs

[a] smodels answer set finding time + probability computing time

[b] partial grounding time + probability computing time

[c] mrf creating time + sampling time

– the rules

$$c\!=\!v \leftarrow B', pf^{c}_{\Box,r}\!=\!v, \textit{not Assigned}_r.$$

for all $v \in \textit{Dom}(c)$, where B' is the body of the random selection rule r. These rules assign v to c when the uniform distribution applies to $c\!=\!v$.

Example 5 continued *The simple P-log program Π in Example 5 can be turned into the following multi-valued probabilistic program. In addition to $\tau(\Pi)$ we have*

$$\begin{array}{c}
0.25 : pf^{Roll(d)}_{O(d)=M,r(d)}\!=\!6 \mid 0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}\!=\!5 \mid \\
0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}\!=\!4 \mid 0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}\!=\!3 \mid \\
0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}\!=\!2 \mid 0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}\!=\!1 \\
\frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}\!=\!6 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}\!=\!5 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}\!=\!4 \mid \\
\frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}\!=\!3 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}\!=\!2 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}\!=\!1 \\
\textit{Roll}(d)\!=\!x \leftarrow \textit{Owner}(d)\!=\!\textit{Mike}, pf^{Roll(d)}_{O(d)=M,r(d)}\!=\!x, \\
\qquad\qquad \textit{not Intervene}(\textit{Roll}(d)) \\
\textit{Assigned}_{r(d)} \leftarrow \textit{Owner}(d)\!=\!\textit{Mike}, \textit{not Intervene}(\textit{Roll}(d)) \\
\textit{Roll}(d)\!=\!x \leftarrow pf^{Roll(d)}_{\Box,r(d)}\!=\!x, \textit{not Assigned}_{r(d)}.
\end{array}$$

Theorem 5 *For any consistent simple P-log program Π of signature σ and any possible world W of Π, we construct a formula F_W as follows.*

$$\begin{array}{rl}
F_W = & (\bigwedge_{c=v\in W} c\!=\!v) \wedge \\
& (\bigwedge_{\substack{c,v\,:\\ c\,=\,v \textit{ is possible in } W,\\ W \models c\,=\,v \textit{ and } PR_W(c) \neq \emptyset}} pf^{c}_{B_{W,c},r_{W,c}}\!=\!v) \\
& \wedge(\bigwedge_{\substack{c,v\,:\\ c\,=\,v \textit{ is possible in } W,\\ W \models c\,=\,v \textit{ and } PR_W(c) = \emptyset}} pf^{c}_{\Box,r_{W,c}}\!=\!v)
\end{array}$$

We have

$$\mu_\Pi(W) = P_{\Pi^{\rm LP^{MLN}}}(F_W),$$

and, for any proposition A of signature σ,

$$P_\Pi(A) = P_{\Pi^{\rm LP^{MLN}}}(A).$$

Example 5 continued *For the possible world*

$$\begin{array}{rl}
W = & \{\textit{Roll}(D_1)\!=\!6, \textit{Roll}(D_2)\!=\!3, \textit{Even}(D_1), \sim\!\textit{Even}(D_2), \\
& \textit{Owner}(D_1)\!=\!\textit{Mike}, \textit{Owner}(D_2)\!=\!\textit{John}\},
\end{array}$$

F_W is

$$\begin{array}{l}
\textit{Roll}(D_1)\!=\!6 \wedge \textit{Roll}(D_2)\!=\!3 \wedge \textit{Even}(D_1) \wedge \sim\!\textit{Even}(D_2) \\
\wedge\ \textit{Owner}(D_1)\!=\!\textit{Mike} \wedge \textit{Owner}(D_2)\!=\!\textit{John} \\
\wedge\ pf^{Roll(D_1)}_{O(D_1)=M,r}\!=\!6 \wedge pf^{Roll(D_2)}_{\Box,r}\!=\!3.
\end{array}$$

It can be seen that $\hat{\mu}_\Pi(W) = \frac{1}{4} \times \frac{1}{6} = P_{\Pi^{\rm LP^{MLN}}}(F_W)$.

The embedding tells us that the exact inference in simple P-log is no harder than the one in LP$^{\rm MLN}$.

Experiments

Following the translation described above, it is possible to compute a tight P-log program by translating it to LP$^{\rm MLN}$, and further turn that into the MLN instance following the translation introduced in Section 5, and then compute it using an MLN solver.

Table 1 shows the performance comparison between this method and the native P-log implementation on some examples, which are modified from the ones from (Baral, Gelfond, and Rushton 2009). P-log 1.0.0 (`http://www.depts.ttu.edu/cs/research/krlab/plog.php`) implements two algorithms. The first algorithm (plog1) translates a P-log program to an ASP program and uses ASP solver SMODELS to find all possible worlds of the P-log program. The second algorithm (plog2) produces a partially ground P-log program relevant to the query, and evaluates partial possible worlds to compute the probability of formulas. ALCHEMY 2.0 implements several algorithms for inference and learning. Here we use MC-SAT for lazy probabilistic inference, which combines MCMC with satisfiability testing. ALCHEMY first creates Markov Random Field (MRF) and then perform MC-SAT on the MRF created. The default setting of ALCHEMY performs 1000 steps sampling. We also tested with 5000 steps sampling to produce probability that is very close to the true probability. The experiments were performed on an Intel Core2 Duo CPU E7600 3.06GH with 4GB RAM running Ubuntu 13.10. The timeout was for 10 minutes.

The experiments showed the clear advantage of the translation method that uses ALCHEMY. It is more scalable, and can be tuned to yield more precise probability with more sampling or less precise but fast computation, by changing sampling parameters. The P-log implementation of the second algorithm led to segment faults in many cases.

9 Other Related Work

We observed that ProbLog can be viewed as a special case of LP^{MLN}. This result can be extended to embed Logic Programs with Annotated Disjunctions (LPAD) in LP^{MLN} based on the fact that any LPAD program can be further turned into a ProbLog program by eliminating disjunctions in the heads (Gutmann 2011, Section 3.3).

It is known that LPAD is related to several other languages. In (Vennekens et al. 2004), it is shown that Poole's ICL (Poole 1997) can be viewed as LPAD, and that acyclic LPAD programs can be turned into ICL. This indirectly tells us how ICL is related to LP^{MLN}.

CP-logic (Vennekens, Denecker, and Bruynooghe 2009) is a probabilistic extension of FO(ID) (Denecker and Ternovska 2007) that is closely related to LPAD.

PrASP (Nickles and Mileo 2014) is another probabilistic ASP language. Like P-log and LP^{MLN}, probability distribution is defined over stable models, but the weights there directly represent probabilities.

Similar to LP^{MLN}, log-linear description logics (Niepert, Noessner, and Stuckenschmidt 2011) follow the weight scheme of log-linear models in the context of description logics.

10 Conclusion

Adopting the log-linear models of MLN, language LP^{MLN} provides a simple and intuitive way to incorporate the concept of weights into the stable model semantics. While MLN is an undirected approach, LP^{MLN} is a directed approach, where the directionality comes from the stable model semantics. This makes LP^{MLN} closer to P-log and ProbLog. On the other hand, the weight scheme adopted in LP^{MLN} makes it amenable to apply the statistical inference methods developed for MLN computation. More work needs to be done to find how the methods studied in machine learning will help us to compute weighted stable models. While a fragment of LP^{MLN} can be computed by existing implementations of and MLNs, one may design a native computation method for the general case.

The way that we associate weights to stable models is orthogonal to the way the stable model semantics are extended in a deterministic way. Thus it is rather straightforward to extend LP^{MLN} to allow other advanced features, such as aggregates, intensional functions and generalized quantifiers.

Acknowledgements We are grateful to Michael Gelfond for many useful discussions regarding the different ideas behind P-log and LP^{MLN}, and to Evgenii Balai, Michael Bartholomew, Amelia Harrison, Yunsong Meng, and the anonymous referees for their useful comments. This work was partially supported by the National Science Foundation under Grants IIS-1319794 and IIS-1526301, and ICT R&D program of MSIP/IITP 10044494 (WiseKB).

References

Balduccini, M., and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In *International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, 9–18.

Baral, C.; Gelfond, M.; and Rushton, J. N. 2009. Probabilistic reasoning with answer sets. *TPLP* 9(1):57–144.

Bauters, K.; Schockaert, S.; De Cock, M.; and Vermeir, D. 2010. Possibilistic answer set programming revisited. In *26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*.

Buccafurri, F.; Leone, N.; and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *Knowledge and Data Engineering, IEEE Transactions on* 12(5):845–860.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, 2462–2467.

Denecker, M., and Ternovska, E. 2007. Inductive situation calculus. *Artificial Intelligence* 171(5-6):332–360.

Erdem, E., and Lifschitz, V. 2003. Tight logic programs. *TPLP* 3:499–518.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. *TPLP* 15(03):358–401.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R., and Bowen, K., eds., *Proceedings of International Logic Programming Conference and Symposium*, 1070–1080. MIT Press.

Gutmann, B. 2011. *On Continuous Distributions and Parameter Estimation in Probabilistic Logic Programs*. Ph.D. Dissertation, KU Leuven.

Lee, J., and Lifschitz, V. 2003. Loop formulas for disjunctive logic programs. In *Proceedings of International Conference on Logic Programming (ICLP)*, 451–465.

Lee, J., and Wang, Y. 2015. A probabilistic extension of the stable model semantics. In *International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2015 Spring Symposium Series*.

Lee, J.; Meng, Y.; and Wang, Y. 2015. Markov logic style weighted rules under the stable model semantics. In Technical Communications of the 31st International Conference on Logic Programming.

Lin, F., and Zhao, Y. 2004. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157:115–137.

Nickles, M., and Mileo, A. 2014. Probabilistic inductive logic programming based on answer set programming. In *15th International Workshop on Non-Monotonic Reasoning (NMR 2014)*.

Niepert, M.; Noessner, J.; and Stuckenschmidt, H. 2011. Log-linear description logics. In *IJCAI*, 2153–2158.

Poole, D. 1997. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence* 94:7–56.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62(1-2):107–136.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP)*, 715–729.

Vennekens, J.; Verbaeten, S.; Bruynooghe, M.; and A, C. 2004. Logic programs with annotated disjunctions. In *Proceedings of International Conference on Logic Programming (ICLP)*, 431–445.

Vennekens, J.; Denecker, M.; and Bruynooghe, M. 2009. CP-logic: A language of causal probabilistic events and its relation to logic programming. *TPLP* 9(3):245–308.