Statistical Relational Extension of Answer Set Programming

Joohyung Lee[1 [0000−0002−9569−5575]] and Zhun Yang[1 [0000−0002−9043−5774]]

Arizona State University, Tempe AZ 85287, USA
{joolee,zyang90}@asu.edu

Keywords: Answer Set Programming · Statistical Relational Learning

Abstract. This tutorial presents a statistical relational extension of the answer set programming language called LP$^{\text{MLN}}$, which incorporates the concept of weighted rules into the stable model semantics following the log-linear models of Markov Logic. An LP$^{\text{MLN}}$ program defines a probability distribution over "soft" stable models, which may not satisfy all rules, but the more rules with larger weights they satisfy, the higher their probabilities, thus allowing for an intuitive and elaboration tolerant representation of problems that require both logical and probabilistic reasoning. The extension provides a natural way to overcome the deterministic nature of the stable model semantics, such as resolving inconsistencies in answer set programs, associating probability to stable models, and applying statistical inference and learning with probabilistic stable models. We also present formal relations between LP$^{\text{MLN}}$ and other related formalisms, which produce ways of performing inference and learning in LP$^{\text{MLN}}$.

1 Introduction

Answer Set Programming (ASP) [23], based on the stable model semantics [12], is a widely used knowledge representation framework that facilitates elegant and efficient representations for many problem domains that require complex reasoning. However, ASP has no built-in mechanism to represent probabilistic uncertainty and to statistically induce knowledge from the data.

This tutorial presents an extension of ASP called LP$^{\text{MLN}}$ that incorporates the concept of weighted rules following the log-linear models of Markov Logic [27]. Instead of the classical logic adopted in Markov Logic, language LP$^{\text{MLN}}$ adopts stable models as the logical component. The relationship between LP$^{\text{MLN}}$ and Markov Logic is analogous to the known relationship between ASP and SAT.

For example, consider the simple ASP knowledge base KB_1:

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

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^{MLN}.

This paper is organized as follows. After reviewing the stable model semantics in Section 2, we introduce the syntax and semantics of the language LP^{MLN} in Section 3. Section 4 relates LP^{MLN} to other formalisms. Section 5 explains how to learn weights of LP^{MLN} programs from data, and Section 6 presents an implementation based on these ideas. Section 7 presents a fragment of LP^{MLN} that is simpler and has been used in defining a probabilistic action language [20] and a neural ASP extension [33].

The tutorial is based on previous publications about LP^{MLN} [19, 22, 18, 21, 33].

2 Review: Stable Model Semantics

The stable model semantics was defined for rules of a simple form [12] to arbitrary first-order formulas [9].

We assume a first-order signature σ that contains no function constants of positive arity, which yields finitely many Hedrbrand interpretations.[1] The syntax of formulas is defined the same as in the standard first-order logic. We say that a formula is *negative* if every occurrence of every atom in this formula is in the scope of negation.

In this tutorial, for simplicity, we mainly consider a *rule* of the form

$$A \leftarrow B \wedge N \tag{1}$$

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 a comma for conjunction, a semi-colon for disjunction, *not* for negation, as widely used in the literature on logic programming. For example, N could be

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

[1] It is straightforward to extend LP^{MLN} to allow function constants of positive arity as long as the program is finitely groundable.

which can be also written as

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

where each B_i is an atom.

We write $\{A_1\}^{\mathrm{ch}} \leftarrow Body$, where A_1 is an atom, to denote the rule $A_1 \leftarrow Body \wedge \neg\neg A_1$. 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 under the stable model semantics (a.k.a. *answer set 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.

A *weak constraint* [5, 6] has the form

$$:\sim F \quad [Weight\ @\ Level]$$

where F is a conjunction of literals, *Weight* is a real number, and *Level* is a nonnegative integer.

Let Π be a program $\Pi_1 \cup \Pi_2$, where Π_1 is an answer set program that does not contain weak constraints, and Π_2 is a set of ground weak constraints. We call I a *stable model* of Π if it is a stable model of the standard program Π_1. For every stable model I of Π and any nonnegative integer l, the *penalty* of I at level l, denoted by $Penalty_\Pi(I, l)$, is defined as

$$\sum_{\substack{:\sim\ F\,[w@l] \in \Pi_2, \\ I \models F}} w.$$

For any two stable models I and I' of Π, we say I is *dominated* by I' if

- there is some nonnegative integer l such that $Penalty_\Pi(I', l) < Penalty_\Pi(I, l)$ and
- for all integers $k > l$, $Penalty_\Pi(I', k) = Penalty_\Pi(I, k)$.

A stable model of Π is called *optimal* if it is not dominated by another stable model of Π.

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

LP$^{\mathrm{MLN}}$ is a probabilistic logic programming language that extends answer set programs with the concept of weighted rules, whose weight scheme is adopted from that of Markov Logic. In this section, we introduce the syntax and semantics of LP$^{\mathrm{MLN}}$, and show how LP$^{\mathrm{MLN}}$ can be used to resolve certain inconsistencies in ASP programs or express different certainty levels with the weighting scheme.

3.1 Syntax of LP$^{\mathbf{MLN}}$

The syntax of LP$^{\mathrm{MLN}}$ defines a set of weighted rules. More precisely, an 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.

Similar to answer set programs, we say that an LP$^{\mathrm{MLN}}$ program is *ground* if its rules contain no variables. We identify any LP$^{\mathrm{MLN}}$ program Π of signature σ with a ground 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., $\overline{\Pi} = \{R \mid w : R \in \Pi\}$.

3.2 Semantics of LP$^{\mathbf{MLN}}$ (Reward-Based)

A stable model of an 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. 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 ([19]) *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 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 SM[Π] 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

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

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

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

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

We omit the subscript Π if the context is clear. We say that I is a *(probabilistic) stable model* of Π if $P_\Pi(I) \neq 0$. Intuitively, $P_\Pi(I)$ indicates how likely it is to draw I as a stable model of some maximal subset of $\overline{\Pi}$.

For any proposition A, probability $P_\Pi(A)$ 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 only interested in stable models that satisfy all hard rules (because hard rules represent 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 $\mathrm{LP}^{\mathrm{MLN}}$ 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 SM′[Π] be the set

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

and let

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

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

Notice the absence of $\lim\limits_{\alpha\to\infty}$ in the definition of $P'_\Pi[I]$. Also, unlike $P_\Pi(I)$, SM′[Π] may be empty, in which case $P'_\Pi(I)$ is undefined. 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 ([19]) *If* SM′[Π] *is not empty, for every interpretation* I, $P'_\Pi(I)$ *coincides with* $P_\Pi(I)$.

It follows that if SM′[Π] is not empty, then every stable model of Π (with non-zero probability) should satisfy all hard rules in Π.

3.3 Examples

The weighting scheme of LP$^{\text{MLN}}$ provides a simple and effective way to resolve certain inconsistencies in ASP programs.

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

$$\begin{array}{lll} KB_1 & \alpha : \mathit{Bird}(x) \leftarrow \mathit{ResidentBird}(x) & (r_1) \\ & \alpha : \mathit{Bird}(x) \leftarrow \mathit{MigratoryBird}(x) & (r_2) \\ & \alpha : \leftarrow \mathit{ResidentBird}(x), \mathit{MigratoryBird}(x) & (r_3) \\ KB_2 & \alpha : \mathit{ResidentBird}(\mathit{Jo}) & (r_4) \\ KB_3 & \alpha : \mathit{MigratoryBird}(\mathit{Jo}) & (r_5) \end{array}$$

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

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

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

$$\begin{array}{lll} KB'_2 & 2 : \mathit{ResidentBird}(\mathit{Jo}) & (r'_4) \\ KB'_3 & 1 : \mathit{MigratoryBird}(\mathit{Jo}) & (r'_5) \end{array}$$

Then the table for $KB_1 \cup KB'_2 \cup KB'_3$ is 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

$P(Bird(Jo)) = (e^2 + e^1)/(e^2 + e^1 + e^0) = 0.67 + 0.24$, 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.

Example 2. "Markov Logic has the drawback that it cannot express (non-ground) inductive definitions" [10] because it relies on classical models. This is not the case with LP^{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.

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

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*)

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

Thus we get

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

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 a lower probability than the first two.

3.4 Alternative Formulation (Penalty-Based)

In the definition of the LP$^{\text{MLN}}$ semantics in Section 3.2, the weight of a stable model I is computed from all ground rules $w : R$ such that $I \models R$. Alternatively, the semantics can be reformulated in a "penalty" based way. More precisely, we define the penalty-based weight of an interpretation I as the exponentiated negative sum of the weights of the rules that are not satisfied by I (when I is a stable model of $\overline{\Pi_I}$). Let

$$W_\Pi^{\text{pnt}}(I) = \begin{cases} exp\left(- \sum\limits_{w:R \,\in\, \Pi \text{ and } I \not\models R} w \right) & \text{if } I \in \text{SM}[\Pi]; \\ 0 & \text{otherwise} \end{cases}$$

and

$$P_\Pi^{\text{pnt}}(I) = \lim_{\alpha \to \infty} \frac{W_\Pi^{\text{pnt}}(I)}{\sum\limits_{J \in \text{SM}[\Pi]} W_\Pi^{\text{pnt}}(J)}.$$

The following theorem tells us that the LP$^{\text{MLN}}$ semantics can be equivalently reformulated using the concept of penalty-based weights.

Theorem 1 ([18]) *For any* LP$^{\text{MLN}}$ *program* Π *and any interpretation* I*,*

$$W_\Pi(I) \propto W_\Pi^{\text{pnt}}(I) \qquad \textit{and} \qquad P_\Pi(I) = P_\Pi^{\text{pnt}}(I).$$

Although the penalty-based reformulation appears to be slightly more complicated, it has a few desirable features. One of them is that adding a rule that is trivially true does not affect the penalty-based weight of an interpretation, which is not the case with the reward-based weight.

More importantly, this reformulation leads to a better translation of LP$^{\text{MLN}}$ programs into answer set programs as will be shown in Section 4.1.

Example 3. Consider the LP$^{\text{MLN}}$ program Π in Example 1.

$$\begin{array}{ll} \alpha : \mathit{Bird}(\mathit{Jo}) \leftarrow \mathit{ResidentBird}(\mathit{Jo}) & (r_1) \\ \alpha : \mathit{Bird}(\mathit{Jo}) \leftarrow \mathit{MigratoryBird}(\mathit{Jo}) & (r_2) \\ \alpha : \bot \leftarrow \mathit{ResidentBird}(\mathit{Jo}), \mathit{MigratoryBird}(\mathit{Jo}) & (r_3) \\ 2 : \mathit{ResidentBird}(\mathit{Jo}) & (r_4') \\ 1 : \mathit{MigratoryBird}(\mathit{Jo}) & (r_5') \end{array}$$

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

I	Π_I	$W_\Pi^{\text{pnt}}(I)$	$P_\Pi^{\text{pnt}}(I)$
$\emptyset$	$\{r_1, r_2, r_3\}$	e^{-3}	$\frac{e^{-3}}{e^{-1}+e^{-2}+e^{-3}}$
$\{R(Jo)\}$	$\{r_2, r_3, r_4'\}$	$e^{-\alpha-1}$	0
$\{M(Jo)\}$	$\{r_1, r_3, r_5'\}$	$e^{-\alpha-2}$	0
$\{B(Jo)\}$	$\{r_1, r_2, r_3\}$	0	0
$\{R(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_4'\}$	e^{-1}	$\frac{e^{-1}}{e^{-1}+e^{-2}+e^{-3}}$
$\{M(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_5'\}$	e^{-2}	$\frac{e^{-2}}{e^{-1}+e^{-2}+e^{-3}}$
$\{R(Jo), M(Jo)\}$	$\{r_4', r_5'\}$	$e^{-3\alpha}$	0
$\{R(Jo), M(Jo), B(Jo)\}$	$\{r_1, r_2, r_4', r_5'\}$	$e^{-\alpha}$	0

Note that for each interpretation, its probability computed with penalty-based weights is exactly the same as the one computed with reward-based weights.

4 Relation to Other Languages

This section relates LP^{MLN} to ASP, MLN, and ProbLog. The translation helps us to compute LP^{MLN} using the tools of the related formalisms.

4.1 Relating LP^{MLN} to ASP

An answer set program can be turned into an LP^{MLN} program by assigning the infinite weight to every rule. That is, for any answer set program $\mathcal{P} = \{R_1, \ldots, R_n\}$, the corresponding LP^{MLN} program $\Pi_\mathcal{P}$ is $\{\alpha : R_1, \ldots, \alpha : R_n\}$.

The following theorem establishes how ASP can be viewed as a fragment of LP^{MLN}.

Theorem 2 ([19]) *For any answer set program $\mathcal{P}$, the (deterministic) stable models of $\mathcal{P}$ are exactly the (probabilistic) stable models of $\Pi_\mathcal{P}$ whose weight is $e^{k\alpha}$, where k is the number of all (ground) rules in $\mathcal{P}$. If $\mathcal{P}$ has at least one stable model, then all stable models of $\Pi_\mathcal{P}$ have the same probability and are thus the stable models of $\mathcal{P}$ as well.*

The other direction, turning an LP^{MLN} program into an answer set program, is possible with the help of weak constraints.

For any ground LP^{MLN} program Π, the translation lpmln2asp(Π) is obtained from Π by turning each weighted rule

$$w_i : Head_i \leftarrow Body_i$$

into

$$\begin{aligned} unsat(i) &\leftarrow Body_i, not\ Head_i \\ Head_i &\leftarrow Body_i, not\ unsat(i) \\ :\sim\ unsat(i) &\quad [w_i@l] \end{aligned}$$

where $unsat(i)$ is a new atom, and $w_i@l$ is 1@1 (denoting penalty 1 at a higher priority) if w_i is α and is $w_i@0$ (denoting penalty w_i at a lower priority) otherwise.

In the case when $Head_i$ is $\bot$, the translation can be further simplified: we simply turn $w_i : \bot \leftarrow Body_i$ into $\quad :\sim Body_i \quad [w_i@l]$.

The following theorem allows for computing LP$^{\mathrm{MLN}}$ using ASP solvers.

Theorem 3 ([22]) *For any* LP$^{\mathrm{MLN}}$ *program* Π*, there is a 1-1 correspondence* ϕ *between the most probable stable models of* Π *and the optimal stable models of* lpmln2asp(Π)*, where* $\phi(I) = I \cup \{unsat(i) \mid w_i : F_i \in \Pi,\ I \not\models F_i\}$.

Example 4. For the LP$^{\mathrm{MLN}}$ program Π with 4 soft rules:

$$\begin{aligned} 10 &: \ q \leftarrow p \\ 1 &: \ r \leftarrow p \\ 5 &: \ p \\ -20 &: \ \bot \leftarrow \neg r \end{aligned}$$

SM[Π] has 5 elements: $\emptyset$, $\{p\}$, $\{p, q\}$, $\{p, r\}$, $\{p, q, r\}$. Among them, $\{p, q\}$ is the most probable stable model, whose penalty-based weight is e^{19}, while $\{p, q, r\}$ is a probabilistic stable model whose penalty-based weight is e^0. The translation lpmln2asp(Π) yields

$$\begin{array}{l|l|l} unsat(1) \leftarrow p, not\ q & q \leftarrow p, not\ unsat(1) & :\sim unsat(1) \quad [10@0] \\ unsat(2) \leftarrow p, not\ r & r \leftarrow p, not\ unsat(2) & :\sim unsat(2) \quad [1@0] \\ unsat(3) \leftarrow not\ p & p \leftarrow not\ unsat(3) & :\sim unsat(3) \quad [5@0] \\ unsat(4) \leftarrow not\ r & \bot \leftarrow not\ r, not\ unsat(4) & :\sim unsat(4)\ [-20@0] \end{array}$$

whose optimal stable model is $\{p, q, unsat(2), unsat(4)\}$ with the penalty at level 0 being $1 - 20 = -19$, while $\{p, q, r\}$ is a stable model whose penalty at level 0 is 0. Equivalently, the fourth rule in Π can be simply translated to

$$:\sim not\ r \quad [-20@0].$$

The following example illustrates how the translation accounts for the difference between hard rules and soft rules by assigning different penalty levels.

Example 5. Consider the LP$^{\mathrm{MLN}}$ program Π in Example 1.

$$\begin{aligned} &\alpha : Bird(Jo) \leftarrow ResidentBird(Jo) && (r_1) \\ &\alpha : Bird(Jo) \leftarrow MigratoryBird(Jo) && (r_2) \\ &\alpha : \bot \leftarrow ResidentBird(Jo), MigratoryBird(Jo) && (r_3) \\ &2 : ResidentBird(Jo) && (r_4) \\ &1 : MigratoryBird(Jo) && (r_5) \end{aligned}$$

Π has three probabilistic stable models: $\emptyset$, $\{Bird(Jo), ResidentBird(Jo)\}$, and $\{Bird(Jo), MigratoryBird(Jo)\}$, all of which satisfy all the hard rules r_1, r_2, and r_3. Among these three probabilistic stable models, $\{Bird(Jo), ResidentBird(Jo)\}$ is the most probable stable model. The translation lpmln2asp(Π) yields

$$\begin{array}{l|l} unsat(1) \leftarrow ResidentBird(Jo), not\ Bird(Jo) & Bird(Jo) \leftarrow ResidentBird(Jo), not\ unsat(1) \\ unsat(2) \leftarrow MigratoryBird(Jo), not\ Bird(Jo) & Bird(Jo) \leftarrow MigratoryBird(Jo), not\ unsat(2) \\ unsat(4) \leftarrow not\ ResidentBird(Jo) & ResidentBird(Jo) \leftarrow not\ unsat(4) \\ unsat(5) \leftarrow not\ MigratoryBird(Jo) & MigratoryBird(Jo) \leftarrow not\ unsat(5) \end{array}$$

$$
\begin{array}{ll}
:\sim\ unsat(1) & [1@1] \\
:\sim\ unsat(2) & [1@1] \\
:\sim\ ResidentBird(Jo), MigratoryBird(Jo) & [1@1] \\
:\sim\ unsat(4) & [2@0] \\
:\sim\ unsat(5) & [1@0]
\end{array}
$$

whose stable models with penalty 0 at level 1 are 1-1 correspondent to 3 probabilistic stable models of Π. Among these 3 stable models, $\{Bird(Jo), ResidentBird(Jo), unsat(5)\}$ has the least penalty (i.e., 1) at level 0, thus is the optimal stable model of lpmln2asp(Π).

In some applications, one may not want any hard rules to be violated, assuming that hard rules encode definite knowledge. For that, lpmln2asp(Π) can be modified by simply turning hard rules into the usual ASP rules. Then the stable models of lpmln2asp(Π) satisfy all hard rules. For example, the LP$^{\mathrm{MLN}}$ program in Example 5 can be translated into the following ASP program.

$$
\begin{array}{ll}
Bird(Jo) \leftarrow ResidentBird(Jo) & \\
Bird(Jo) \leftarrow MigratoryBird(Jo) & \\
\bot \leftarrow ResidentBird(Jo), MigratoryBird(Jo) & \\
 & \\
unsat(4) \leftarrow not\ ResidentBird(Jo) & \\
ResidentBird(Jo) \leftarrow not\ unsat(4) & \\
:\sim\ unsat(4) & [2@0] \\
 & \\
unsat(5) \leftarrow not\ MigratoryBird(Jo) & \\
MigratoryBird(Jo) \leftarrow not\ unsat(5) & \\
:\sim\ unsat(5) & [1@0]
\end{array}
$$

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

This section relates LP$^{\mathrm{MLN}}$ to Markov Logic Networks (MLNs).

Embedding MLNs in LP$^{\mathrm{MLN}}$ Similar to the way that SAT can be embedded in ASP, Markov Logic can be easily embedded in LP$^{\mathrm{MLN}}$. More precisely, any MLN $\mathbb{L}$ of signature σ can be turned into an 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.

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 4 ([19]) *Any MLN $\mathbb{L}$ and its* LP$^{\mathrm{MLN}}$ *representation $\Pi_{\mathbb{L}}$ have the same probability distribution over all interpretations.*

The embedding tells us that the exact inference in LP^{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 LP^{MLN} are non-disjunctive, counting the stable models of LP^{MLN} is in #P, which yields that the exact inference for non-disjunctive LP^{MLN} programs is #P-complete. Therefore, approximation algorithms, such as Gibbs sampling, may be desirable for computing large LP^{MLN} programs.

It follows from Theorem 3 and Theorem 4 that Maximum A Posteriori (MAP) inference in MLN can also be reduced to the optimal stable model finding of an ASP program. For any ground MLN program $\mathbb{L}$, the translation mln2asp($\mathbb{L}$) is obtained from $\mathbb{L}$ by turning each weighted formula $w : F$ into the weak constraint $:\sim \neg F \ \ [w@0]$ and adding a choice rule $\{A\}^{\text{ch}}$ for every ground atom A of σ. [2]

Theorem 5 ([22]) *For any Markov Logic Network Π, the most probable models of Π are precisely the optimal stable models of the ASP program with weak constraints* mln2asp(Π).

Similarly, MAP inference in ProbLog and Pearl's Causal Models can be reduced to finding an optimal stable model of a program with weak constraints in view of the reduction of ProbLog to LP^{MLN} (Theorem 4 from [19]) and the reduction of Causal Models to LP^{MLN} (Theorem 4 from [15]) thereby allowing us to apply combinatorial optimization methods in standard ASP solvers to these languages.

Completion: Turning LP^{MLN} to MLN We say that LP^{MLN} program Π is *tight* if the unweighted program $\overline{\Pi}$ is tight according to [16], i.e., the positive dependency graph of $\overline{\Pi}$ is acyclic. It is known that the stable models of a tight logic program coincide with the models of the program's completion [8]. This yielded a way to compute stable models using SAT solvers. The method can be extended to tight LP^{MLN} programs so that queries involving probabilistic stable models can be computed using existing implementations of MLNs, such as Alchemy. [3]

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

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

for each ground atom A. This is a straightforward extension of the completion from [16] by simply assigning the infinite weight α to the completion formulas.

Theorem 6 ([19]) *For any tight* LP^{MLN} *program Π such that* $\text{SM}'[\Pi]$ *is not empty, Π (under the* LP^{MLN} *semantics) and $Comp(\Pi)$ (under the MLN semantics) have the same probability distribution over all interpretations.*

[2] F in the weak constraint here is an arbitrary formula, that is more general than the form reviewed in Section 2. We can use a tool like F2LP [17] to turn formulas into the input language of CLINGO.

[3] http://alchemy.cs.washington.edu

The theorem can be generalized to non-tight programs by considering loop formulas [24], which we skip here for brevity.

4.3 Relating LP$^{\mathbf{MLN}}$ to ProbLog

It turns out that LP$^{\mathrm{MLN}}$ is a proper generalization of ProbLog, a well-developed probabilistic logic programming language that is based on the distribution semantics by [28].

Review of ProbLog The review follows [10]. 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* $\mathcal{P}$ is a tuple $\langle PF, \Pi \rangle$, where

- PF is a set of ground probabilistic facts of the form $pr :: a$ where pr is a real number in $[0, 1]$ and a is a probabilistic atom,
- Π is a set of ground rules of the following form

$$A \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$$

where A, $B_1, \ldots B_n$ are atoms from σ $(0 \leq m \leq n)$, 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 $TC = \{a_1, \ldots, a_m\}$, the *probability* of a total choice TC, denoted $Pr_{\mathcal{P}}(TC)$, 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 $b_1, \ldots, b_n$ 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 $\mathcal{P} = \langle PF, \Pi \rangle$ such that $\Pi \cup TC$ has a "total" (two-valued) well-founded model for each total choice TC. Given such $\mathcal{P}$, the probability of an interpretation I, denoted $P_{\mathcal{P}}(I)$, is defined as $Pr_{\mathcal{P}}(TC)$ if there exists a total choice TC such that I is the total well-founded model of $\Pi \cup TC$, and 0 otherwise.

ProbLog as a Special Case of LP$^{\mathbf{MLN}}$ Given a ProbLog program $\mathcal{P} = \langle PF, \Pi \rangle$, we construct the corresponding LP$^{\mathrm{MLN}}$ program $\mathcal{P}'$ as follows:

- For each probabilistic fact $pr :: a$ in $\mathcal{P}$, LP$^{\mathrm{MLN}}$ program $\mathcal{P}'$ contains
  - $ln(pr) : a$ and $ln(1-pr) : \ \leftarrow a$ if $0 < pr < 1$;
  - $\alpha : a$ if $pr = 1$;

- $\alpha: \leftarrow a$ if $pr = 0$.

– For each rule $R \in \Pi$, $\mathcal{P}'$ contains $\alpha : R$. In other words, R is identified with a hard rule in $\mathcal{P}'$.

Theorem 7 ([19]) *Any well-defined ProbLog program $\mathcal{P}$ and its* LP$^{\text{MLN}}$ *representation $\mathcal{P}'$ have the same probability distribution over all interpretations.*

Example 6. Consider the ProbLog program

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

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

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

Syntactically, LP$^{\text{MLN}}$ 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, LP$^{\text{MLN}}$ allows rules to be weighted as well as facts and does not distinguish between probabilistic facts and derived atoms. Semantically, while the ProbLog semantics is based on well-founded models, LP$^{\text{MLN}}$ 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, LP$^{\text{MLN}}$ can handle multiple stable models in a flexible way similar to the way MLN handles multiple models.

5 Weight Learning

In this section, we present the concept of weight learning in LP$^{\text{MLN}}$ and learning algorithms for LP$^{\text{MLN}}$.

A parameterized LP$^{\text{MLN}}$ program $\hat{\Pi}$ is defined similarly to an LP$^{\text{MLN}}$ program Π except for that non-α weights (i.e., "soft" weights) are replaced with distinct parameters to be learned. By $\hat{\Pi}(\mathbf{w})$, where $\mathbf{w}$ is a list of real numbers whose length is the same as the number of soft rules, we denote the LP$^{\text{MLN}}$ program obtained from $\hat{\Pi}$ by replacing the parameters with $\mathbf{w}$. The weight learning task for a parameterized LP$^{\text{MLN}}$ program is to find the MLE (Maximum Likelihood Estimation) of the parameters as in Markov Logic. Formally, given a parameterized LP$^{\text{MLN}}$ program $\hat{\Pi}$ and a ground formula O (often in the form of conjunctions of literals) called *observation* or *training data*, the LP$^{\text{MLN}}$ parameter learning task is to find the values $\mathbf{w}$ of parameters such that the probability of O under the LP$^{\text{MLN}}$ program Π is maximized. In other words, the learning task is to find

$$\operatorname*{argmax}_{\mathbf{w}} P_{\hat{\Pi}(\mathbf{w})}(O). \tag{2}$$

5.1 Gradient Method for Learning Weights From a Complete Stable Model

Same as in Markov Logic, there is no closed-form solution for (2), but the gradient ascent method can be applied to find the optimal weights in an iterative manner.

We first compute the gradient. Given a (non-ground) LP$^{\text{MLN}}$ program Π whose SM[Π] is non-empty and given a stable model I of Π, the base-e logarithm of $P_\Pi(I)$, $lnP_\Pi(I)$, is

$$-\sum_{w_i:R_i\in\Pi^{\text{soft}}} w_i n_i(I) - ln\sum_{J\in\text{SM}[\Pi]} exp\Big(-\sum_{w_i:R_i\in\Pi^{\text{soft}}} w_i n_i(J)\Big).$$

The partial derivative of $lnP_\Pi(I)$ w.r.t. $w_i(\neq\alpha)$ is

$$\begin{aligned}\frac{\partial lnP_\Pi(I)}{\partial w_i} &= -n_i(I) + \frac{\sum_{J\in\text{SM}[\Pi]} exp(-\sum_{w_i:R_i\in\Pi^{\text{soft}}} w_i n_i(J))n_i(J)}{\sum_{K\in\text{SM}[\Pi]} exp(-\sum_{w_i:R_i\in\Pi^{\text{soft}}} w_i n_i(K))}\\ &= -n_i(I) + \sum_{J\in\text{SM}[\Pi]}\left(\frac{exp(-\sum_{w_i:R_i\in\Pi^{\text{soft}}} w_i n_i(J))}{\sum_{K\in\text{SM}[\Pi]} exp(-\sum_{w_i:R_i\in\Pi^{\text{soft}}} w_i n_i(K))}\right)n_i(J)\\ &= -n_i(I) + \sum_{J\in\text{SM}[\Pi]} P_\Pi(J)n_i(J) = -n_i(I) + \underset{J\in\text{SM}[\Pi]}{E}[n_i(J)]\end{aligned}$$

where $\underset{J\in\text{SM}[\Pi]}{E}[n_i(J)] = \sum_{J\in\text{SM}[\Pi]} P_\Pi(J)n_i(J)$ is the expected number of false ground rules obtained from R_i.

Since the log-likelihood above is a concave function of the weights, any local maximum is a global maximum, and maximizing $P_\Pi(I)$ can be done by the standard gradient ascent method by updating each weight w_i by $w_i+\lambda\cdot(-n_i(I)+\underset{J\in\text{SM}[\Pi]}{E}[n_i(J)])$ until it converges.[4]

However, similar to Markov Logic, computing $\underset{J\in\text{SM}[\Pi]}{E}[n_i(J)]$ is intractable [27]. In the next section, we turn to an MCMC sampling method to find its approximate value.

5.2 Sampling Method: MC-ASP

The following is an MCMC algorithm for LP$^{\text{MLN}}$, which adapts the algorithm MC-SAT for Markov Logic [26] by considering the penalty-based reformulation and by using an ASP solver instead of a SAT solver for sampling.

When all the weights w_i of soft rules are nonpositive, $1-e^{w_i}$ (in step (b)) is in the range $[0,1)$, and thus it validly represents a probability. At each iteration,

[4] Note that although any local maximum is a global maximum for the log-likelihood function, there can be multiple combinations of weights that achieve the maximum probability of the training data.

Algorithm 1 MC-ASP

Input: An LP$^{\text{MLN}}$ program Π whose soft rules' weights are non-positive and a positive integer N.
Output: Samples $I^1, \ldots, I^N$

1. Choose a (probabilistic) stable model I^0 of Π.
2. Repeat the following for $j = 1, \ldots, N$
   (a) $M \leftarrow \emptyset$;
   (b) For each ground instance of each rule $w_i : R_i \in \Pi^{\text{soft}}$ that is false in I^{j-1}, add the ground instance to M with probability $1 - e^{w_i}$;
   (c) Randomly choose a (probabilistic) stable model I^j of Π that satisfies no rules in M.

the sample is chosen from stable models of Π, and, consequently, it must satisfy all hard rules. For soft rules, the higher its weight, the less likely it is to be included in M, and thus less likely to be not satisfied by the sample generated from M.

The following theorem states that MC-ASP satisfies the MCMC criteria of ergodicity and detailed balance, which justifies the soundness of the algorithm.

Theorem 8 ([21]) *The Markov chain generated by MC-ASP satisfies ergodicity and detailed balance.*[5]

Steps 1 and 2(c) of the algorithm require finding a (probabilistic) stable model of LP$^{\text{MLN}}$, which can be computed by an ASP solver CLINGO using the translation lpmln2asp(Π) in Section 4.1. Step 2(c) also requires a uniform sampler for answer sets, which can be computed by a system like XORRO [11].

Algorithm 2 is a weight learning algorithm for LP$^{\text{MLN}}$ based on gradient ascent using MC-ASP (Algorithm 1) for collecting samples. Step 2(b) of MC-ASP requires that w_i be non-positive in order for $1 - e^{w_i}$ to represent a probability. Unlike in the Markov Logic setting, converting positive weights into non-positive weights cannot be done in LP$^{\text{MLN}}$ simply by replacing $w : F$ with $-w : \neg F$, due to the difference in the FOL and the stable model semantics. Algorithm 2 converts Π into an equivalent program Π^{neg} whose rules' weights are non-positive, before calling MC-ASP. The following theorem justifies the soundness of this method.[6]

Theorem 9 ([21]) *When* SM[Π] *is not empty, the program Π^{neg} specifies the same probability distribution as the program Π.*

[5] A Markov chain is *ergodic* if there is a number m such that any state can be reached from any other state in any number of steps greater than or equal to m.

Detailed balance means $P_\Pi(X)Q(X \to Y) = P_\Pi(Y)Q(Y \to X)$ for any samples X and Y, where $Q(X \to Y)$ denotes the probability that the next sample is Y given that the current sample is X.

[6] Note that Π^{neg} is only used in MC-ASP. The output of Algorithm 2 may have positive weights.

Algorithm 2 Algorithm for learning weights using LPMLN2ASP

Input: Π: A parameterized LP^{MLN} program in the input language of LPMLN2ASP; O: A stable model represented as a set of constraints (that is, $\leftarrow$ *not* A is in O if a ground atom A is true; $\leftarrow A$ is in O if A is not true); δ: a fixed real number to be used for the terminating condition.
Output: Π with learned weights.
Process:

1. Initialize the weights of soft rules $R_1, \ldots, R_m$ with some initial weights $\mathbf{w}^0$.
2. Repeat the following for $j = 1, \ldots$ until $max\{|w_i^j - w_i^{j-1}| : i = 1, \ldots, m\} < \delta$:
   (a) Compute the stable model of $\Pi \cup O$ using LPMLN2ASP (see below); for each soft rule R_i, compute $n_i(O)$ by counting `unsat` atoms whose first argument is i (i is a rule index).
   (b) Create Π^{neg} by replacing each soft rule R_i of the form $w: \ H(\mathbf{x}) \leftarrow B(\mathbf{x})$ in Π where $w > 0$ with

$$\begin{aligned} &0: \ H(\mathbf{x}) \leftarrow B(\mathbf{x}), \\ &\alpha: \ \texttt{neg}(i, \mathbf{x}) \leftarrow B(\mathbf{x}), \texttt{not } H(\mathbf{x}), \\ &-w: \leftarrow \texttt{not neg}(i, \mathbf{x}). \end{aligned}$$

   (c) Run MC-ASP on Π^{neg} to collect a set S of sample stable models.
   (d) For each soft rule R_i, approximate $\sum_{J \in \text{SM}[\Pi]} P_\Pi(J) n_i(J)$ with $\sum_{J \in S} n_i(J)/|S|$, where n_i is obtained from counting the number of `unsat` atoms whose first argument is i.
   (e) For each $i \in \{1, \ldots, m\}$,
   $w_i^{j+1} \leftarrow w_i^j + \lambda \cdot (-n_i(O) + \sum_{J \in S} n_i(J)/|S|)$.

Non-emptiness of SM[Π] implies that every probabilistic stable model of Π satisfies all hard rules in Π. Thus when SM[Π] is not empty, the probability of a stable model is proportional to its weight accumulated from the soft rules only – the translation from Π to Π^{neg} guarantees that the proportion is preserved for every stable model.

5.3 Extensions

The base case learning in the previous section assumes that the training data is a single stable model and is a complete interpretation. This section extends the framework in a few ways.

Learning from Multiple Stable Models The method described in the previous section allows only one stable model to be used as the training data. Now, suppose we have multiple stable models $I_1, \ldots, I_m$ as the training data. For example, consider the parameterized program $\hat{\Pi}_{coin}$ that describes a coin, which

may or may not land in the head when it is flipped,

$$\begin{aligned} \alpha &: \{flip\} \\ w &: head \leftarrow flip \end{aligned}$$

(the first rule is a choice rule) and three stable models as the training data: $I_1 = \{flip\}$, $I_2 = \{flip\}$, $I_3 = \{flip, head\}$ (the absence of *head* in the answer set is understood as landing in tail), indicating that $\{flip, head\}$ has a frequency of $\frac{1}{3}$, and $\{flip\}$ has a frequency of $\frac{2}{3}$. Intuitively, the more we observe the *head*, the larger the weight of the second rule. Clearly, learning w from only one of I_1, I_2, I_3 won't result in a weight that captures all the three stable models: learning from each of I_1 or I_2 results in the value of w too small for $\{flip, head\}$ to have a frequency of $\frac{1}{3}$ while learning from I_3 results in the value of w too large for $\{flip\}$ to have a frequency of $\frac{2}{3}$.

To utilize the information from multiple stable models, one natural idea is to maximize the joint probability of all the stable models in the training data, which is the product of their probabilities, i.e.,

$$P(I_1, \dots, I_m) = \prod_{j \in \{1,\dots,m\}} P_{\Pi}(I_j).$$

The partial derivative of $lnP(I_1, \dots, I_m)$ w.r.t. $w_i (\neq \alpha)$ is

$$\frac{\partial lnP(I_1, \dots, I_m)}{\partial w_i} = \sum_{j \in \{1,\dots,m\}} \Big(- n_i(I_j) + \mathop{E}_{J \in \mathrm{SM}[\Pi]}[n_i(J)] \Big).$$

In other words, the gradient of the log probability is simply the sum of the gradients of the probability of each stable model in the training data. To update Algorithm 2 to reflect this, we simply repeat step 2(a) to compute $n_i(I_k)$ for each $k \in \{1, \dots, m\}$, and at step 2(e) update w_i as follows:

$$w_i^{j+1} \leftarrow w_i^j + \lambda \cdot \Big(- \sum_{k \in \{1,\dots,m\}} n_i(I_k) + m \cdot \sum_{J \in \mathrm{SM}[\Pi]} P_{\Pi}(J) n_i(J) \Big).$$

Alternatively, learning from multiple stable models can be reduced to learning from a single stable model by introducing one more argument k to every predicate, which represents the index of a stable model in the training data, and rewriting the data to include the index.

Formally, given an $\mathrm{LP}^{\mathrm{MLN}}$ program Π and a set of its stable models $I_1, \dots, I_m$, let Π^m be an $\mathrm{LP}^{\mathrm{MLN}}$ program obtained from Π by appending one more argument k to the list of arguments of every predicate that occurs in Π, where k is a schematic variable that ranges over $\{1, \dots, m\}$. Let

$$I = \bigcup_{i \in \{1,\dots,m\}} \{p(\mathbf{t}, i) \mid p(\mathbf{t}) \in I_i\}. \tag{3}$$

The following theorem asserts that the weights of the rules in Π that are learned from the multiple stable models $I_1, \dots, I_m$ are identical to the weights of the rules in Π^m that are learned from the single stable model I that conjoins $\{I_1, \dots, I_m\}$ as in (3).

Theorem 10 ([21]) *For any parameterized* $\mathrm{LP}^{\mathrm{MLN}}$ *program* $\hat{\Pi}$*, its stable models* $I_1, \ldots, I_m$ *and* I *as defined as in* (3)*, we have*

$$\operatorname*{argmax}_{\mathbf{w}} P_{\hat{\Pi}^m(\mathbf{w})}(I) = \operatorname*{argmax}_{\mathbf{w}} \prod_{i \in \{1,\ldots,m\}} P_{\hat{\Pi}(\mathbf{w})}(I_i).$$

Example 7. For the program $\hat{\Pi}_{coin}$, to learn from the three stable models I_1, I_2, and I_3 defined before, we consider the program $\hat{\Pi}^3_{coin}$

$$\begin{aligned} \alpha &: \{flip(k)\}. \\ w &: head(k) \leftarrow flip(k). \end{aligned}$$

($k \in \{1,2,3\}$) and combine I_1, I_2, I_3 into one stable model $I = \{flip(1), flip(2), flip(3), head(3)\}$. The weight w in $\hat{\Pi}^3_{coin}$ learned from the single data I is identical to the weight w in $\hat{\Pi}_{coin}$ learned from the three stable models I_1, I_2, I_3.

5.4 Learning in the Presence of Noisy Data

So far, we assumed that the data $I_1, \ldots, I_m$ are (probabilistic) stable models of the parameterized $\mathrm{LP}^{\mathrm{MLN}}$ program. Otherwise, the joint probability would be zero regardless of any weights assigned to the soft rules, and the partial derivative of $lnP(I_1, \ldots, I_m)$ is undefined. However, data gathered from the real world could be noisy, so some data I_i may not necessarily be a stable model. Even then, we still want to learn from the other "correct" instances. We may try to drop noisy data before starting training but enumerating all noisy data could be computationally expensive. Alternatively, we may mitigate the influence of the noisy data by introducing so-called "noise atoms" as follows.

Example 8. Consider again the program $\hat{\Pi}^m_{coin}$. Suppose one of the interpretations I_i in the training data is $\{head(i)\}$. The interpretation is not a stable model of $\hat{\Pi}^m_{coin}$. We obtain $\hat{\Pi}^m_{noisecoin}$ by modifying $\hat{\Pi}^m_{coin}$ to allow for the noisy atom $n(k)$ as follows.

$$\begin{aligned} \alpha &: \{flip(k)\}. \\ w &: head(k) \leftarrow flip(k). \\ \alpha &: head(k) \leftarrow n(k). \\ -u &: n(k). \end{aligned}$$

Here, u is a positive number that is "sufficiently" larger than w. $\{head(i), n(i)\}$ is a stable model of $\hat{\Pi}^m_{noisecoin}$, so that the combined training data I is still a stable model, and thus a meaningful weight w for $\hat{\Pi}^m_{noisecoin}$ can still be learned, given that other "correct" instances I_j ($j \neq i$) dominate in the learning process (as for the noisy example, the corresponding stable model gets a low weight due to the weight assigned to $n(i)$ but not 0).

Furthermore, with the same value of w, the larger u becomes, the closer the probability distribution defined by $\hat{\Pi}^m_{noisecoin}$ approximates the one defined by $\hat{\Pi}^m_{coin}$, so the value of w learned under $\hat{\Pi}^m_{noisecoin}$ approximates the value of w learned under $\hat{\Pi}^m_{coin}$ where the noisy data is dropped.

5.5 Learning from Incomplete Interpretations

In the previous sections, we assume that the training data is given as a (complete) interpretation, i.e., for each atom, it specifies whether it is true or false. In this section, we discuss the general case when the training data is given as a partial interpretation, which omits to specify some atoms to be true or false, or more generally when the training data is in the form of a formula that more than one stable model may satisfy.

Given a non-ground LP$^{\mathrm{MLN}}$ program Π such that SM[Π] is not empty and given a ground formula O as the training data, we have

$$P_{\Pi}(O) = \frac{\sum_{I \models O, I \in \mathrm{SM}[\Pi]} W_{\Pi}(I)}{\sum_{J \in \mathrm{SM}[\Pi]} W_{\Pi}(J)}.$$

The partial derivative of $lnP_{\Pi}(O)$ w.r.t. w_i ($\neq \alpha$) turns out to be

$$\frac{\partial lnP_{\Pi}(O)}{\partial w_i} = - \underset{I \models O, I \in \mathrm{SM}[\Pi]}{E}[n_i(I)] + \underset{J \in \mathrm{SM}[\Pi]}{E}[n_i(J)].$$

It is straightforward to extend Algorithm 2 to reflect the extension. Computing the approximate value of the first term $- \underset{I \models O, I \in \mathrm{SM}[\Pi]}{E}[n_i(I)]$ can be done by sampling on $\Pi^{neg} \cup O$.

6 LP$^{\mathrm{MLN}}$ System

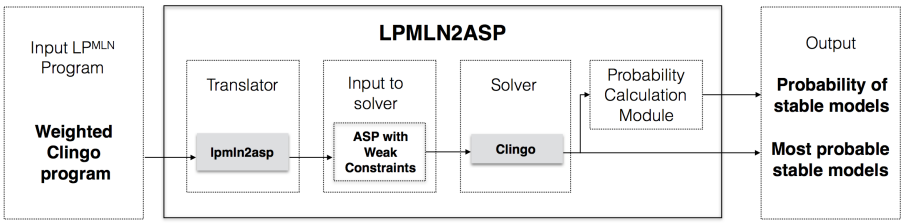

Fig. 1. Architecture of System LPMLN2ASP

System LPMLN2ASP [7] is an implementation of LP$^{\mathrm{MLN}}$ based on the result in Section 3.4 using CLINGO. It can be used for computing the probabilities of stable models, the marginal/conditional probability of a query, as well as the most probable stable models.

[7] https://github.com/azreasoners/lpmln

In the input language of LPMLN2ASP, a soft rule is written in the form

$$w_i \quad Head_i \leftarrow Body_i \tag{4}$$

where w_i is a real number in decimal notation, and $Head_i \leftarrow Body_i$ is a CLINGO rule. A hard rule is written without weights and is identical to a CLINGO rule. For instance, the "Bird" example from [19] can be represented in the input language of LPMLN2ASP as follows. The first three rules represent definite knowledge, while the last two rules represent uncertain knowledge with different confidence.

```
% bird.lpmln
bird(X) :- residentbird(X).
bird(X) :- migratorybird(X).
:- residentbird(X), migratorybird(X).
2 residentbird(jo).
1 migratorybird(jo).
```

The basic command line syntax for executing LPMLN2ASP is

```
lpmln2asp -i <input file> [-r <output file>] [-e <evidence file>]
[-q <query predicates>] [-hr] [-all] [-clingo "<clingo options>"]
```

which follows the syntax of the ALCHEMY command line.

The mode of computation is determined by the options provided to LPMLN2ASP. By default, the system finds a most probable stable model of $\mathsf{lpmln2asp}^{\mathrm{pnt}}(\Pi)$ (MAP estimate) by leveraging CLINGO's a built-in optimization method for weak constraints.

For computing marginal probability, LPMLN2ASP utilizes CLINGO's interface with Python. When CLINGO enumerates each stable model of $\mathsf{lpmln2asp}^{\mathrm{pnt}}(\Pi)$, the computation is interrupted by the *probability computation module*—a Python program which records the stable model and its penalty specified in the `unsat` atoms true in the stable model. Once all the stable models are generated, the control returns to the module, which sums up the recorded penalties to compute the normalization constant and the probability of each stable model. The probabilities of the query atoms (specified by option `-q`) are also calculated by adding the probabilities of the stable models that contain the query atoms. For example, the probability of a query atom `residentbird(jo)` is $\Sigma_{I \models \texttt{residentbird(jo)}} P(I)$. The option `-all` instructs the system to display all stable models and their probabilities.

For conditional probability, the evidence file `<evidence file>` is specified by the option `-e`. The file may contain any CLINGO rules, but usually they are constraints, i.e., rules with the empty head. The main difference from the marginal probability computation is that CLINGO computes $\mathsf{lpmln2asp}^{\mathrm{pnt}}(\Pi) \cup$ `<evidence file>` instead of $\mathsf{lpmln2asp}^{\mathrm{pnt}}(\Pi)$.

Below we illustrate how to use the system for various inferences.

MAP (Maximum A Posteriori) inference: The command line to use is

```
lpmln2asp -i <input file>
```

By default, LPMLN2ASP computes MAP inference. For example, `lpmln2asp -i bird.lpmln` returns

```
residentbird(jo) bird(jo) unsat(5,"1.000000")
Optimization: 1000
OPTIMUM FOUND
```

Marginal probability of all stable models: The command line to use is

```
lpmln2asp -i <input file> -all
```

This mode finds all stable models and calculates their probabilities. For example, `lpmln2asp -i bird.lpmln -all` outputs

```
Answer: 1
residentbird(jo) bird(jo)
unsat(5,"1.000000")
Optimization: 1000
Answer: 2
unsat(4,"2.000000") unsat(5,"1.000000")
Optimization: 3000
Answer: 3
unsat(4,"2.000000") bird(jo)
migratorybird(jo)
Optimization: 2000

Probability of Answer 1 : 0.665240955775
Probability of Answer 2 : 0.0900305731704
Probability of Answer 3 : 0.244728471055
```

Marginal probability of query atoms: The command line to use is

```
lpmln2asp -i <input file> -q <query predicates>
```

This mode calculates the marginal probability of the atoms whose predicates are specified by `-q` option. For example, `lpmln2asp -i birds.lp -q residentbird` outputs

```
residentbird(jo) 0.665240955775
```

Conditional probability of query given evidence: The command line to use is

```
lpmln2asp -i <input file> -q <query predicates> -e <evidence file>
```

This mode computes the conditional probability of a query given the evidence specified in the `<evidence file>`. For example,

```
lpmln2asp -i birds.lp -q residentbird -e evid.db
```

where `evid.db` contains

```
:- not bird(jo).
```

outputs the conditional probability $P(residentbird(X) \mid bird(jo))$:

```
residentbird(jo) 0.73105857863
```

Debugging ASP Programs: The command line to use is

```
lpmln2asp -i <input file> -hr -all
```

By default, LPMLN2ASP does not translate hard rules and passes them to CLINGO as is. The option `-hr` instructs the system to translate hard rules as well. According to Proposition 2 from [19], as long as the LP^{MLN} program has a probabilistic stable model that satisfies all the hard rules, the simpler translation that does not translate hard rules gives the same result as the full translation and is more computationally efficient. Since in many cases hard rules represent definite knowledge that should not be violated, this is desirable.

On the other hand, translating hard rules could be relevant in some other cases, such as debugging an answer set program by finding which rules cause inconsistency. For example, consider a CLINGO input program `bird.lp`, which is similar to `bird.lpmln` but drops the weights in the last two rules. CLINGO finds no stable models for this program. However, if we invoke LPMLN2ASP on the same program as

```
lpmln2asp -i bird.lp -hr
```

the output of LPMLN2ASP shows three probabilistic stable models, each of which shows a way to resolve the inconsistency by ignoring the minimal number of the rules. For instance, one of them is {`bird(jo)`, `residentbird(jo)`}, which disregards the last rule. The other two are similar.

Note that probability computation involves enumerating all stable models, so it can be much more computationally expensive than the default MAP inference. On the other hand, the computation is exact, so compared to an approximate inference, the "gold standard" result is easy to understand. Furthermore, the conditional probability is computed more effectively than the marginal probability because CLINGO effectively prunes many answer sets that do not satisfy the constraints specified in the evidence file.

Computing MLN with LPMLN2ASP A typical example in the MLN literature is a social network domain that describes how smokers influence other people, which can be represented in LP^{MLN} as follows. We assume three people *alice*, *bob*, and *carol*, and assume that *alice* is a smoker, *alice* influences *bob*, *bob* influences *carol*, and nothing else is known.

$$\begin{array}{l} w: \quad smoke(x) \wedge influence(x,y) \rightarrow smoke(y) \\ \alpha: smoke(alice) \qquad \alpha: influence(alice,bob) \qquad \alpha: influence(bob,carol). \end{array} \tag{5}$$

(w is a positive number.) One may expect that *bob* is less likely to be a smoker than *alice*, and *carol* is less likely a smoker than *bob*.

Indeed, the program above defines the following distribution (we omit the *influence* relation, which has a fixed interpretation.)

Possible World	Weight
$\{smoke(alice), \neg smoke(bob), \neg smoke(carol)\}$	$k \cdot e^{8w}$
$\{smoke(alice), smoke(bob), \neg smoke(carol)\}$	$k \cdot e^{8w}$
$\{smoke(bob), \neg smoke(alice), smoke(carol)\}$	0
$\{smoke(alice), smoke(bob), smoke(carol)\}$	$k \cdot e^{9w}$

where $k = e^{3\alpha}$. The normalization constant is the sum of all weights: $k \cdot e^{9w} + 2k \cdot e^{8w}$. This means $P(smoke(alice)) = 1$ and

$$P(smoke(bob)) = \lim_{\alpha \to \infty} \frac{k \cdot e^{8w} + k \cdot e^{9w}}{k \cdot e^{9w} + 2k \cdot e^{8w}} > P(smoke(carol)) = \lim_{\alpha \to \infty} \frac{k \cdot e^{9w}}{k \cdot e^{9w} + 2k \cdot e^{8w}}.$$

The result can be verified by LPMLN2ASP. For $w = 1$, the input program `smoke.lpmln` is

```
1 smoke(Y) :- smoke(X), influence(X, Y).
smoke(alice).    influence(alice, bob).    influence(bob, carol).
```

Executing `lpmln2asp -i smoke.lpmln -q smoke` outputs

```
smoke(alice) 1.00000000000000
smoke(bob) 0.788058442382915
smoke(carol) 0.576116884765829
```

as expected.

On the other hand, if (5) is understood under the MLN semantics (assuming `influence` relation is fixed as before), similar to the above, one can compute

$$P(smoke(bob)) = \frac{e^{8w} + e^{9w}}{3e^{8w} + e^{9w}} = P(smoke(carol)).$$

In other words, the degraded probability along the transitive relation does not hold under the MLN semantics. This is related to the fact that Markov logic cannot express the concept of transitive closure correctly as it inherits the FOL semantics.

According to Theorem 2 in [19], MLN can be easily embedded in LP^{MLN} by adding a choice rule for each atom with an arbitrary weight, similar to the way propositional logic can be embedded in ASP using choice rules. Consequently, it is possible to use system LPMLN2ASP to compute MLN, which is essentially using an ASP solver to compute MLN.

Let `smoke.mln` be the resulting program. Executing `lpmln2asp -i smoke.mln -q smoke` outputs

```
smoke(alice) 1.0
smoke(bob)   0.650244590946
smoke(carol)   0.650244590946
```

which agrees with the computation above.

7 Multi-Valued Probabilistic Programs

In this section, we define a simple fragment of LP^{MLN} that represents probability in a more direct way and can be computed more efficiently. 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 $c{=}v$ [8] 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:

$$p_1 : c{=}v_1 \mid \cdots \mid p_n : c{=}v_n \tag{6}$$

  one for each probabilistic constant c, where $\{v_1, \ldots, v_n\} = Dom(c)$, $v_i \neq v_j$, $0 \leq p_1, \ldots, p_n \leq 1$ 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^{MLN} program $T(\mathbf{\Pi})$ of the same signature as follows.

- For each probabilistic constant declaration (6), $T(\mathbf{\Pi})$ contains, for each $i = 1, \ldots, n$,
  - $ln(p_i) : c{=}v_i$ if $0 < p_i < 1$;
  - $\alpha :\ c{=}v_i$ if $p_i = 1$;
  - $\alpha : \leftarrow c{=}v_i$ if $p_i = 0$.
- For each rule in Π of form (1), $T(\mathbf{\Pi})$ contains

$$\alpha :\quad A \leftarrow B, N.$$

- For each constant c, $T(\mathbf{\Pi})$ contains the uniqueness of value constraints

$$\alpha : \bot \leftarrow c{=}v_1 \wedge c = v_2 \tag{7}$$

  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

$$\alpha : \bot \leftarrow \neg \bigvee_{v \in Dom(c)} c{=}v\ . \tag{8}$$

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

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 9. The multi-valued probabilistic program

$$\begin{aligned}&0.25 : Outcome{=}6 \mid 0.15 : Outcome{=}5\\ &\quad \mid 0.15 : Outcome{=}4 \mid 0.15 : Outcome{=}3\\ &\qquad \mid 0.15 : Outcome{=}2 \mid 0.15 : Outcome{=}1\\ &Win \leftarrow Outcome{=}6.\end{aligned}$$

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

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

We say an interpretation of $\mathbf{\Pi}$ is *consistent* if it satisfies the hard rules (7) for every constant and (8) for every probabilistic constant. For any consistent interpretation I, we define the set $TC(I)$ ("Total Choice") to be $\{c = v \mid c$ is a probabilistic constant such that $c = v \in I\}$ and define

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

For any interpretation I, we define

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

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 3 ([19]) *For any multi-valued probabilistic program* $\mathbf{\Pi}$ *such that each* p_i *in* (6) *is positive for every probabilistic constant* c*, if* $\mathrm{SM}''[\mathbf{\Pi}]$ *is not empty, then for any interpretation* I*,* $P''_{\mathbf{\Pi}}(I)$ *coincides with* $P_{T(\mathbf{\Pi})}(I)$*.*

MVPP has been used in extending action language $\mathcal{BC}+$ [2, 3] to a probabilistic manner [20, 31, 30], and as a basis of NeurASP to extend probabilistic answer set programs to embrace neural networks [33].

8 Conclusion

There are more works on LP$^{\text{MLN}}$ that we do not discuss here. Strong equivalence between LP$^{\text{MLN}}$ programs was studied in [25]. Parallel LP$^{\text{MLN}}$ solver appeared in [32]. Splitting theorem for LP$^{\text{MLN}}$ program is studied in [29]. PCM can be embedded in LP$^{\text{MLN}}$ [15]. The relationship between LP$^{\text{MLN}}$ and P-Log was shown in [4].

LP$^{\text{MLN}}$ has served as the basis of or link between other formalisms. A decision-theoretic extension of LP$^{\text{MLN}}$ (DT-LP$^{\text{MLN}}$) was presented in [30] where a utility measure is associated to each (probabilistic) stable model in addition to its probability. Based on this, a translation from $p\mathcal{BC}+$ to Partially Observable Markov Decision Processes (POMDPs) was designed [31] where LP$^{\text{MLN}}$ is used to generate components of POMDP, including states, actions, transitions, and their probabilities. [14] extended event calculus with probabilistic reasoning empowered by LP$^{\text{MLN}}$. Recently, a new system PLINGO [13] was presented as a probabilistic extension of CLINGO where LP$^{\text{MLN}}$ serves as a middle-ground formalism connecting available input languages of PLINGO, including LP$^{\text{MLN}}$, P-log, ProbLog, and ASP.

LP$^{\text{MLN}}$ has been applied in many domains to model probabilistic inference. [7] showed that the classification of objects in an image could be improved by considering their semantic context, where LP$^{\text{MLN}}$ is used to model both constraints and probabilistic information provided by the classifier. [1] designed an automated and interpretable fact-checking system where uncertain rules and facts extracted from knowledge graphs and web documents are modeled by LP$^{\text{MLN}}$.

We conclude this tutorial by inviting the readers to explore further along this line of research. One particular promising direction is with the notion of probability, one can link soft models of logic programs with neural networks [33], which helps avoid some issues with a pure deep learning approach.

Acknowledgements This work was partially supported by the National Science Foundation under Grants IIS-1815337 and IIS-2006747.

References

1. Ahmadi, N., Lee, J., Papotti, P., Saeed, M.: Explainable fact checking with probabilistic answer set programming. In: Conference for Truth and Trust Online (2019)
2. Babb, J., Lee, J.: Action language $\mathcal{BC}+$: Preliminary report. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2015)
3. Babb, J., Lee, J.: Action language $\mathcal{BC}+$. Journal of Logic and Computation **30**(4), 899–922 (2020)
4. Balai, E., Gelfond, M.: On the relationship between P-log and LPMLN. In: IJCAI (2016)
5. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive datalog by constraints. IEEE Transactions on Knowledge and Data Engineering **12**(5), 845–860 (2000)
6. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2: Input language format. ASP Standardization Working Group, Tech. Rep (2012)

7. Eiter, T., Kaminski, T.: Exploiting contextual knowledge for hybrid classification of visual objects. In: European Conference on Logics in Artificial Intelligence. pp. 223–239. Springer (2016)
8. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3**, 499–518 (2003)
9. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence **175**, 236–263 (2011)
10. Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory and Practice of Logic Programming **15**(03), 358–401 (2015)
11. Gebser, M., Schaub, T., Marius, S., Thiele, S.: xorro: Near uniform sampling of answer sets by means of xor (2016), https://potassco.org/labs/2016/09/20/xorro.html
12. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Proceedings of International Logic Programming Conference and Symposium. pp. 1070–1080. MIT Press (1988)
13. Hahn, S., Janhunen, T., Kaminski, R., Romero, J., Rühling, N., Schaub, T.: Plingo: A system for probabilistic reasoning in clingo based on lp mln. In: Rules and Reasoning: 6th International Joint Conference on Rules and Reasoning, RuleML+ RR 2022, Berlin, Germany, September 26–28, 2022, Proceedings. pp. 54–62. Springer (2022)
14. Katzouris, N., Artikis, A.: Woled: A tool for online learning weighted answer set rules for temporal reasoning under uncertainty. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning. vol. 17, pp. 790–799 (2020)
15. Lee, J., Meng, Y., Wang, Y.: Markov logic style weighted rules under the stable model semantics. In: CEUR Workshop Proceedings. vol. 1433 (01 2015)
16. Lee, J., Lifschitz, V.: Loop formulas for disjunctive logic programs. In: Proceedings of International Conference on Logic Programming (ICLP). pp. 451–465 (2003)
17. Lee, J., Palla, R.: System F2LP – computing answer sets of first-order formulas. In: Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 515–521 (2009)
18. Lee, J., Talsania, S., Wang, Y.: Computing LPMLN using ASP and MLN solvers. Theory and Practice of Logic Programming (2017). https://doi.org/10.1017/S1471068417000400
19. Lee, J., Wang, Y.: Weighted rules under the stable model semantics. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 145–154 (2016)
20. Lee, J., Wang, Y.: A probabilistic extension of action language $\mathcal{BC}+$. Theory and Practice of Logic Programming **18(3–4)**, 607–622 (2018)
21. Lee, J., Wang, Y.: Weight learning in a probabilistic extension of answer set programs. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 22–31 (2018)
22. Lee, J., Yang, Z.: LPMLN, weak constraints, and P-log. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 1170–1177 (2017)
23. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2**, 526–541 (2001)
24. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**, 115–137 (2004)

25. Luo, M., Lee, J.: Strong equivalence for LPMLN programs. In: ICLP (Technical Communications) (2019)
26. Poon, H., Domingos, P.: Sound and efficient inference with probabilistic and deterministic dependencies. In: AAAI. vol. 6, pp. 458–463 (2006)
27. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1-2), 107–136 (2006)
28. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP). pp. 715–729 (1995)
29. Wang, B., Zhang, Z., Xu, H., Shen, J.: Splitting an LPMLN program. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (1) (2018)
30. Wang, Y., Lee, J.: Elaboration tolerant representation of markov decision process via decision-theoretic extension of probabilistic action language pBC+. In: International Conference on Logic Programming and Nonmonotonic Reasoning. pp. 224–238. Springer (2019)
31. Wang, Y., Zhang, S., Lee, J.: Bridging commonsense reasoning and probabilistic planning via a probabilistic action language. Theory and Practice of Logic Programming **19**(5-6), 1090–1106 (2019)
32. Wu, W., Xu, H., Zhang, S., Duan, J., Wang, B., Zhang, Z., He, C., Zong, S.: LPMLNModels: A parallel solver for LPMLN. In: 2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI). pp. 794–799. IEEE (2018)
33. Yang, Z., Ishay, A., Lee, J.: NeurASP: Embracing neural networks into answer set programming. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI). pp. 1755–1762 (2020). https://doi.org/10.24963/ijcai.2020/243