A Probabilistic Extension of 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 present a probabilistic extension of logic programs under the stable model semantics, inspired by the idea of Markov Logic Networks. The proposed language, called LP$^{\mathrm{MLN}}$, is a generalization of logic programs under the stable model semantics, and as such, embraces the rich body of research in knowledge representation. The language is also a generalization of ProbLog, and is closely related to Markov Logic Networks, which implies that the computation can be carried out by the techniques developed for them. LP$^{\mathrm{MLN}}$ appears to be a natural language for probabilistic answer set programming, and as an example we show how an elaboration tolerant representation of transition systems in answer set programs can be naturally extended to the probabilistic setting.

Introduction

Logic programs under the stable model semantics (Gelfond and Lifschitz 1988) is the language of Answer Set Programming (ASP). Many useful knowledge representation constructs have been introduced in ASP, and several efficient ASP solvers are available. However, like many other logical approaches, ASP is not well suited for handling uncertainty.

A Markov Logic Network (MLN) (Richardson and Domingos 2006) is a successful approach to combine first-order logic and a probabilistic graphical model in a single representation. There, each formula is associated with a weight, and the probability distribution over possible worlds is derived from the weights of the formulas that are satisfied by the possible worlds. Like ASP, there are a few implementations of MLN. However, the logical component of MLN is the standard first-order logic, which has difficulty in representing defaults, causal relations, inductive definitions, and aggregates, that ASP handles well.

We introduce a simple approach to combine the two successful formalisms—logic programs under the stable model semantics and Markov Logic Networks—without sacrificing each of their strengths in logical and probabilistic reasoning. The proposed language, called LP$^{\mathrm{MLN}}$, is a generalization of logic programs under the stable model semantics, and as such, embraces the rich body of research in knowledge representation developed in answer set programming.

MLN can be easily embedded in LP$^{\mathrm{MLN}}$, and the other direction of embedding is also possible via the concept of loop formulas, which is similar to the reduction of answer set programs to classical propositional logic (Lin and Zhao 2004; Lee 2005). The reduction allows us to use an implementation of MLN to compute LP$^{\mathrm{MLN}}$ under certain conditions, similar to the way ASP programs can be computed by SAT solvers.

LP$^{\mathrm{MLN}}$ is also a kind of probabilistic logic programming language. In particular, we show that LP$^{\mathrm{MLN}}$ generalizes ProbLog (Raedt, Kimmig, and Toivonen 2007; Fierens et al. 2013). While ProbLog semantics is based on well-founded models, LP$^{\mathrm{MLN}}$ handles stable model reasoning for more general classes of programs.

LP$^{\mathrm{MLN}}$ appears to be a natural basis for probabilistic answer set programming, towards shifting the current logic-based foundation of ASP to the novel foundation that embraces probabilistic reasoning. As an example we show how elaboration tolerant representations of transition systems in answer set programs can be naturally extended to the probabilistic setting.

The paper is organized as follows. After reviewing the stable model semantics and Markov Logic Networks, we show how they can be merged resulting in LP$^{\mathrm{MLN}}$. Then we show how LP$^{\mathrm{MLN}}$ is related to ASP, MLN, and ProbLog, and how it can be applied to representing probabilistic transition systems.

Preliminaries

Throughout this paper, we assume a finite first-order signature σ that contains no function constants of positive arity. There are finitely many Herbrand interpretations of σ, each of which is finite as well.

Review: Stable Model Semantics

A *rule* over signature σ is of the form

$$A_1; \dots; A_k \leftarrow A_{k+1}, \dots A_m, \mathit{not}\ A_{m+1}, \dots, \mathit{not}\ A_n, \mathit{not\ not}\ A_{n+1}, \dots, \mathit{not\ not}\ A_p \tag{1}$$

($0 \leq k \leq m \leq n \leq p$) where all A_i are atoms of σ possibly containing variables.[1] We will often identify (1) with the

[1] Double negations are useful for encoding choice rules.

implication (written backward):

$$A_1 \vee \cdots \vee A_k \leftarrow A_{k+1} \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \wedge \neg\neg A_{n+1} \wedge \cdots \wedge \neg\neg A_p \ . \quad (2)$$

A *logic program* is a finite set 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 (2) in Π. 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 "$A_1 \vee \cdots \vee A_k \leftarrow A_{k+1} \wedge \cdots \wedge A_m$" for all rules (2) in Π such that $I \models \neg A_{m+1} \wedge \cdots \wedge \neg A_n \wedge \neg\neg A_{n+1} \wedge \cdots \wedge \neg\neg A_p$. The Herbrand interpretation I is called a *stable model* of Π (denoted by $I \models_{\text{SM}} \Pi$) if I is a minimal Herbrand model of Π^I.[2] For example, for the program

$$\begin{array}{ll} p \leftarrow q & p \leftarrow \mathit{not}\ r \\ q \leftarrow p & r \leftarrow \mathit{not}\ p \end{array} \quad (3)$$

the stable models are $\{p, q\}$ and $\{r\}$. The reduct relative to $\{p, q\}$ is $\{p \leftarrow q. \quad q \leftarrow p. \quad p.\}$, for which $\{p, q\}$ is the minimal model; the reduct relative to $\{r\}$ is $\{p \leftarrow q. \quad q \leftarrow p. \quad r.\}$, for which $\{r\}$ is the minimal model.

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.

Review: Markov Logic Networks

The following is a review of MLN from (Richardson and Domingos 2006), slightly reformulated in order to facilitate our discussion.

A *Markov Logic Network (MLN)* $\mathbb{L}$ of signature σ is a finite set of pairs $\langle F, w\rangle$ (also written as a "weighted formula" $w : F$), where F is a first-order formula of σ and w is either a real number or a symbol α denoting the "hard weight." We say that an MLN is *ground* if its formulas contain no variables.

We first define the semantics for ground MLNs. For any ground MLN $\mathbb{L}$ of signature σ and any Herbrand interpretation I of σ, we define $\mathbb{L}_I$ to be the set of formulas in $\mathbb{L}$ that are satisfied by I. The *weight* of an interpretation I under $\mathbb{L}$, denoted $W_{\mathbb{L}}(I)$, is defined as

$$W_{\mathbb{L}}(I) = exp\left(\sum_{\substack{w:F \in \mathbb{L} \\ F \in \mathbb{L}_I}} w\right).$$

The probability of I under $\mathbb{L}$, denoted $Pr_{\mathbb{L}}[I]$, is defined as

$$Pr_{\mathbb{L}}[I] = \lim_{\alpha\to\infty} \frac{W_{\mathbb{L}}(I)}{\sum_{J\in PW} W_{\mathbb{L}}(J)},$$

where PW ("Possible Worlds") is the set of all Herbrand interpretations of σ. We say that I is a *model* of $\mathbb{L}$ if $Pr_{\mathbb{L}}[I] \neq 0$.

[2] Minimality is in terms of set inclusion. We identify an Herbrand interpretation with the set of atoms that are true in it.

The basic idea of MLN is to allow formulas to be soft constrained, where a model does not have to satisfy all formulas, but is associated with the weight that is contributed by the satisfied formulas. For every interpretation (i.e., possible world) I, there is a unique maximal subset of formulas in the MLN that I satisfies. Obviously, this subset is $\mathbb{L}_I$, and the weight of I is obtained from the weights of those "contributing" formulas in $\mathbb{L}_I$. An interpretation that does not satisfy certain formulas receives "penalties" because such formulas do not contribute to the weight of that interpretation.

The definition is extended to any non-ground MLN by identifying it with its *ground instance*. Any MLN $\mathbb{L}$ of signature σ can be identified with the ground MLN, denoted $gr_\sigma[\mathbb{L}]$, by turning each formula in $\mathbb{L}$ into a set of ground formulas as described in (Richardson and Domingos 2006, Table II). The weight of each ground formula in $gr_\sigma[\mathbb{L}]$ is the same as the weight of the formula in $\mathbb{L}$ from which it is obtained.

Language LP^{MLN}

Syntax of LP^{MLN}

The syntax of LP^{MLN} defines a set of weighted rules, which can be viewed as a special case of the syntax of MLN by identifying rules with implications. More precisely, an LP^{MLN} program $\mathbb{P}$ is a finite set of pairs $\langle R, w\rangle$ (also written as a weighted rule $w : R$), where R is a rule of the form (1) and w is either a real number or a symbol α for the "hard weight."

We say that an LP^{MLN} program is *ground* if its rules contain no variables. We identify any LP^{MLN} program $\mathbb{P}$ of signature σ with a ground LP^{MLN} program $gr_\sigma[\mathbb{P}]$, whose rules are obtained from the rules of $\mathbb{P}$ by replacing every variable with every ground term of σ. The weight of a ground rule in $gr_\sigma[\mathbb{P}]$ is the same as the weight of the rule in $\mathbb{P}$ from which it is obtained.

We define $\Pi_{\mathbb{P}}$ to be the logic program obtained from $\mathbb{P}$ by disregarding weights, i.e., $\Pi_{\mathbb{P}} = \{R \mid w : R \in \mathbb{P}\}$.

Semantics of LP^{MLN}

For any ground LP^{MLN} program $\mathbb{P}$ of signature σ and any Herbrand interpretation I of σ, we define $\mathbb{P}_I$ to be the set of rules in $\mathbb{P}$ which are satisfied by I. As in MLN, the weight of the interpretation is obtained from the weights of those "contributing" formulas. The weight of I, denoted by $W_{\mathbb{P}}(I)$, is defined as

$$W_{\mathbb{P}}(I) = exp\left(\sum_{\substack{w:R \in \mathbb{P} \\ R \in \mathbb{P}_I}} w\right)$$

if I is a stable model of $\mathbb{P}_I$; otherwise $W_{\mathbb{P}}(I) = 0$. The probability of I under $\mathbb{P}$, denoted $Pr_{\mathbb{P}}[I]$, is defined as

$$Pr_{\mathbb{P}}[I] = \lim_{\alpha\to\infty} \frac{W_{\mathbb{P}}(I)}{\sum_{J\in PW} W_{\mathbb{P}}(J)}$$

where PW is the set of all Herbrand interpretations of σ. We say that I is a *(probabilistic) stable model* of $\mathbb{P}$ if $Pr_{\mathbb{P}}[I] \neq 0$.

The intuition here is similar to that of MLN. For each possible world I, we try to find a maximal subset of $\Pi_{\mathbb{P}}$ 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 $\Pi_{\mathbb{P}}$, and the probability distribution is over these stable models. Unlike MLN, such a subset may not necessarily exist, which means that no subset can account for the stability of the model. In that case, since we are interested in stable models only, the weight of the interpretation is assigned 0. In the other case, it does not seem obvious that there is a *unique* maximal subset that accounts for the stability of I. Nevertheless, it follows from the following proposition that this is indeed the case, and the unique maximal subset is exactly $\mathbb{P}_I$.

Proposition 1 *For any 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.

Example 1 *Consider an* LP$^{\mathrm{MLN}}$ *program $\mathbb{P}$:*

$$1:\ p \leftarrow q \quad (r_1) \qquad\qquad 2:\ p \leftarrow \mathit{not}\ r \quad (r_3)$$
$$1:\ q \leftarrow p \quad (r_2) \qquad\qquad 3:\ r \leftarrow \mathit{not}\ p. \quad (r_4)$$

$\Pi_{\mathbb{P}}$ *is the same as* (3)*. The weight and the probability of each interpretation are shown in the following table, where Z_1 is $e^2+e^6+2e^7$, and Z_2 is $e+e^2+3e^6+3e^7$.*

I	$\mathbb{P}_I$	$Pr_{\mathbb{P}}[I]$
$\emptyset$	$\{r_1, r_2\}$	e^2/Z_1
$\{p\}$	$\{r_1, r_3, r_4\}$	e^6/Z_1
$\{q\}$	$\{r_2\}$	0
$\{r\}$	$\{r_1, r_2, r_3, r_4\}$	e^7/Z_1
$\{p, q\}$	$\{r_1, r_2, r_3, r_4\}$	e^7/Z_1
$\{q, r\}$	$\{r_2, r_3, r_4\}$	0
$\{p, r\}$	$\{r_1, r_3, r_4\}$	0
$\{p, q, r\}$	$\{r_1, r_2, r_3, r_4\}$	0

The stable models $\{p, q\}$ and $\{r\}$ of $\Pi_{\mathbb{P}}$ are the stable models of $\mathbb{P}$ with the highest probability. In addition, $\mathbb{P}$ has two other stable models, which do not satisfy some rules in $\Pi_{\mathbb{P}}$.

It is easy to observe the following facts.

Proposition 2 *Let $\mathbb{P}$ be an* LP$^{\mathrm{MLN}}$ *program.*

- *Every (probabilistic) stable model of $\mathbb{P}$ is an (MLN) model of $\mathbb{P}$.*
- *Every stable model of $\Pi_{\mathbb{P}}$ is a (probabilistic) stable model of $\mathbb{P}$.*

In each bullet, the reverse direction does not hold as the example above illustrates.

Example 2 *Fierens et al. (2013) remark that "Markov Logic has the drawback that it cannot express (non-ground) inductive definitions." This limitation does not apply to* LP$^{\mathrm{MLN}}$ *as it adopts the stable model semantics in place of the standard first-order logic semantics as in MLN. For instance, the following* LP$^{\mathrm{MLN}}$ *program correctly describes the probabilities of paths, which are induced by the probabilities of the edges that participate in forming the paths.*

$$\begin{array}{ll} w_1: & Edge(1,2) \\ w_2: & Edge(2,3) \\ \cdots & \\ \alpha: & Path(x,y) \leftarrow Edge(x,y) \\ \alpha: & Path(x,y) \leftarrow Path(x,z), Path(z,y). \end{array}$$

Relation to ASP, MLN and ProbLog

Relation to ASP

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

Theorem 1 *For any logic program Π that has at least one stable model, the stable models of Π and the (probabilistic) stable models of* LP$^{\mathrm{MLN}}$ *program $\mathbb{P}_\Pi$ coincide, and all stable models of $\mathbb{P}_\Pi$ have the same probability.*

Theorem 1 does not hold when Π has no stable model. For example, consider $\Pi = \{\leftarrow \mathit{not}\ p\}$, which has no stable model. On the other hand, $\mathbb{P}_\Pi$ is $\{\alpha :\ \leftarrow \mathit{not}\ p\}$, and has the stable model $\emptyset$ with the probability 1.

The idea of softening rules in 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 answer sets of a program Π plus a set of weak constraints are the answer sets of Π which minimize the penalty calculated from *Weight* and *Level* of violated weak constraints. However, weak constraints are more restricted than weighted rules in LP$^{\mathrm{MLN}}$, and do not have a probabilistic semantics.

Relation to MLN: Embedding MLN in LP$^{\mathrm{MLN}}$

MLN can be easily embedded in LP$^{\mathrm{MLN}}$. More precisely, any MLN $\mathbb{L}$ whose formulas have the form (2) can be turned into an LP$^{\mathrm{MLN}}$ program $\mathbb{P}_\mathbb{L}$ so that the models of $\mathbb{L}$ coincide with the stable models of $\mathbb{P}_\mathbb{L}$. We write $\{H\}^{\mathrm{ch}} \leftarrow \mathit{Body}$ to denote the rule $H \leftarrow \mathit{Body}, \mathit{not\ not}\ H$. This expression is called a *choice rule* in ASP (Lee, Lifschitz, and Palla 2008).

LP$^{\mathrm{MLN}}$ program $\mathbb{P}_\mathbb{L}$ is obtained from $\mathbb{L}$ by adding

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

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

Theorem 2 *For any MLN $\mathbb{L}$ whose formulas have the form* (2)*, $\mathbb{L}$ and $\mathbb{P}_\mathbb{L}$ have the same probability distribution over all interpretations, and consequently, the models of $\mathbb{L}$ and the stable models of $\mathbb{P}_\mathbb{L}$ coincide.*

The rule form restriction imposed in Theorem 2 is not essential. For any MLN $\mathbb{L}$ containing arbitrary formulas, one can turn the formulas in clausal normal form as described in (Richardson and Domingos 2006), and further turn that into the rule form. For instance, $p \vee q \vee \neg r$ is turned into $p \vee q \leftarrow r$.

In accordance with Theorem 2, in Example 1, the models of $\mathbb{P}$ and the stable models of $\mathbb{P} \cup \{w : \{A\}^{ch} \mid A \in \{p, q, r\}\}$ coincide.

Relation to MLN: Turning LP$^{\mathrm{MLN}}$ into MLN

It is known that the stable models of a logic program coincide with the models of a logic program plus all its loop formulas. This allows us to compute the stable models using SAT solvers. The method can be extended to LP$^{\mathrm{MLN}}$ so that their stable models along with the probability distribution can be computed using existing implementations of MLN, such as Alchemy [3] and Tuffy. [4]

Due to lack of space, we refer the reader to (Ferraris, Lee, and Lifschitz 2006) for the definitions of a loop L and a loop formula $LF_{\Pi}(L)$ for program Π consisting of rules of the form (1). The following theorem tells us how the stable model semantics can be reduced to the standard propositional logic semantics, via the concept of loop formulas.

Theorem 3 *(Ferraris, Lee, and Lifschitz 2006) Let Π be a ground logic program, and let X be a set of ground atoms. A model X of Π is a stable model of Π iff, for every loop L of Π, X satisfies $LF_{\Pi}(L)$.*

For instance, program (3) has loops $\{p\}, \{q\}, \{r\}, \{p, q\}$, and the corresponding disjunctive loop formulas are

$$\begin{array}{lcllcl} p & \rightarrow & q \vee \neg r & \qquad r & \rightarrow & \neg p \\ q & \rightarrow & p & \qquad p \wedge q & \rightarrow & \neg r. \end{array} \tag{4}$$

The stable models $\{p, q\}$, $\{r\}$ of (3) are exactly the models of (3) that satisfy (4).

We extend Theorem 3 to turn LP$^{\mathrm{MLN}}$ programs $\mathbb{P}$ into MLN instances. We define $\mathbb{L}_{\mathbb{P}}$ to be the union of $\mathbb{P}$ and $\{\alpha : LF_{\Pi_{\mathbb{P}}}(L) \mid L \text{ is a loop of } \Pi_{\mathbb{P}}\}$.

Theorem 4 *For any LP$^{\mathrm{MLN}}$ program $\mathbb{P}$ such that*

$$\{R \mid \alpha : R \in \mathbb{P}\} \cup \{LF_{\Pi_{\mathbb{P}}}(L) \mid L \textit{ is a loop of } \Pi_{\mathbb{P}}\}$$

is satisfiable, $\mathbb{P}$ and $\mathbb{L}_{\mathbb{P}}$ have the same probability distribution over all interpretations, and consequently, the stable models of $\mathbb{P}$ and the models of $\mathbb{L}_{\mathbb{P}}$ coincide.

Example 3 *When $\mathbb{P}$ is the LP$^{\mathrm{MLN}}$ program in Example 1, $\mathbb{L}_{\mathbb{P}}$ is the union of $\mathbb{P}$ and the loop formulas (4) with the hard weight. One can check that the probability distribution of the models of $\mathbb{L}_{\mathbb{P}}$ under the MLN semantics coincides with the probability distribution of the stable models of $\mathbb{P}$ under the LP$^{\mathrm{MLN}}$ semantics shown in Example 1.*

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

[4] http://http://i.stanford.edu/hazy/hazy/tuffy

In general, it is known that the number of loop formulas blows up (Lifschitz and Razborov 2006). As LP$^{\mathrm{MLN}}$ is a generalization of logic programs under the stable model semantics, this blow-up is unavoidable in the context of LP$^{\mathrm{MLN}}$ as well, as illustrated by Example 2. This calls for a better computational method such as the incremental addition of loop formulas as in ASSAT (Lin and Zhao 2004).

In the special case when the program is *tight* (that is, its dependency graph is acyclic (Lee 2005)), the size of loop formulas is linear in the size of input programs. Later in this paper, we formalize a few examples in tight LP$^{\mathrm{MLN}}$ programs, and compute their probabilistic stable models using Alchemy.

Relation to ProbLog

Interestingly, it turns out that LP$^{\mathrm{MLN}}$ is a proper generalization of ProbLog, a prominent approach in probabilistic logic programming that is based on the distribution semantics by Sato (1995).

Review: ProbLog The review follows (Fierens et al. 2013). 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 $\langle PF, \Pi \rangle$, 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, and
- Π is a set of ground rules of the form (1) such that $k = 1$ and $p = n$, and the head does not contain a probabilistic atom.

Probabilistic atoms act as random variables and are assumed to be independent from each other. A *total choice* C is any subset of the probabilistic atoms. Given a total choice $C = \{a_1, \ldots, a_m\}$, the *probability of a total choice* C, denoted $Pr_{\mathbb{P}}[C]$, is defined as

$$Pr_{\mathbb{P}}[C] = pr(a_1) \times \cdots \times pr(a_m) \times (1 - pr(b_1)) \times \cdots \times (1 - pr(b_n))$$

where $b_1, \ldots, b_n$ are probabilistic atoms not belonging to C, 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 $\mathbb{P} = \langle PF, \Pi \rangle$ such that $C \cup \Pi$ has a "total" (two-valued) well-founded model for each possible total choice C. Given such $\mathbb{P}$, for each interpretation I, $Pr_{\mathbb{P}}[I]$ is defined as $Pr_{\mathbb{P}}[C]$ if there exists a total choice C such that I is the total well-founded model of $C \cup \Pi$, and 0 otherwise.

Example 4 *Consider the ProbLog program:*

$$\begin{array}{ll} 0.6 :: p & \qquad r \leftarrow p \\ 0.4 :: q & \qquad r \leftarrow q \end{array}$$

For each of the 4 total choices, the probability distribution over ProbLog models is as follows.

Total choice	*ProbLog model*	*Probability*
$\emptyset$	$\emptyset$	0.24
$\{p\}$	$\{p, r\}$	0.36
$\{q\}$	$\{q, r\}$	0.16
$\{p, q\}$	$\{p, q, r\}$	0.24

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

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

Theorem 5 *Any well-defined ProbLog program $\mathbb{P}$ and its* LP$^{\mathrm{MLN}}$ *representation $\overline{\mathbb{P}}$ have the same probability distribution over all interpretations.*

Syntactically, 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, 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, LP$^{\mathrm{MLN}}$ handles stable model reasoning for more general classes of programs. Unlike ProbLog where it is only well-defined when each total choice leads to a unique well-founded model, LP$^{\mathrm{MLN}}$ can handle multiple stable models in a flexible way similar to the way MLN handles multiple models. These differences turn out to be essential for the successful LP$^{\mathrm{MLN}}$ representation of probabilistic transition systems in the next section.

Representing Probabilistic Transitions

One of the successful applications of ASP is in conveniently representing transition systems and reasoning about paths in them. However, such a representation does not distinguish which path is more probable than others. We show that by augmenting the known ASP representations of transition systems with weights, the LP$^{\mathrm{MLN}}$ semantics gives an intuitive encoding of the probabilistic transition systems.

Simple Transition

Consider the transition system shown in Figure 1, which has one fluent P and one action A. P is inertial and executing A causes P to be true by some chance. Let λ and $1 - \lambda$ represent the probabilities that the action is successful/unsuccessful. The LP$^{\mathrm{MLN}}$ representation is as follows.

Below i is a schematic variable ranging over integers $\{0, \ldots, m-1\}$. P_i and $\sim P_i$ are atoms representing the (complementary) values of fluent P at time i.[5] A_i is an atom representing if A is executed between time i and $i+1$.

Atoms P_i and $\sim P_i$ have complementary values:

$$\begin{array}{l} \alpha: \ \leftarrow P_i, \sim P_i \\ \alpha: \ \leftarrow \mathit{not}\ P_i, \mathit{not}\ \sim P_i\ . \end{array} \tag{5}$$

[5]Thus $\sim$ is part of a symbol, not a connective.

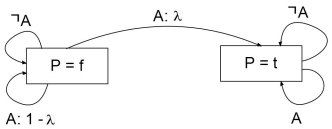

Figure 1: A transition system with probabilistic effects

The effect of A is probabilistic:

$$\alpha: \ P_{i+1} \leftarrow A_i, Aux_i \tag{6}$$

where Aux_i is an auxiliary atom to represent the chance of success, defined as

$$ln(\lambda) : Aux_i \qquad\qquad ln(1-\lambda) : \leftarrow Aux_i\ . \tag{7}$$

The commonsense law of inertia using choice rules:

$$\alpha : \{P_{i+1}\}^{\mathrm{ch}} \leftarrow P_i \qquad\qquad \alpha : \{\sim P_{i+1}\}^{\mathrm{ch}} \leftarrow \sim P_i \tag{8}$$

The execution of A is arbitrary (half a chance to be executed):

$$\alpha: \ \{A_i\}^{\mathrm{ch}}\ . \tag{9}$$

The initial value of P is arbitrary:

$$\alpha : \{P_0\}^{\mathrm{ch}} \qquad\qquad \alpha : \{\sim P_0\}^{\mathrm{ch}}\ . \tag{10}$$

We denote the program consisting of (5)–(10), parameterized with m, by SD_m.

Note that disregarding (7) and dropping Aux_i from (6) yields the standard ASP program representation of the transition system that is similar to Figure 1 where action A has the deterministic effect. That ASP program is exactly the one that is generated from the action language $\mathcal{BC}$+ description of the transition system (Babb and Lee 2015). So LP$^{\mathrm{MLN}}$ gives a natural choice for extending the ASP representation of the transition system in the probabilistic setting.

The following proposition holds for SD_m.

Proposition 3 • *Let $k(I)$ denote the number of Aux_i $(i = 0, \ldots, m-1)$ that are satisfied by I. For any stable model I of SD_m,*

$$Pr_{SD_m}[I] = 0.5^{m+1} \cdot \lambda^{k(I)} \cdot (1-\lambda)^{m-k(I)}.$$

- *For $i = 0, \ldots, m-1$, (i) $Pr_{SD_m}[p_{i+1} \mid \neg p_i, \neg a_i] = 0$. (ii) $Pr_{SD_m}[p_{i+1} \mid \neg p_i, a_i] = \lambda$. (iii) $Pr_{SD_m}[p_{i+1} \mid p_i, \neg a_i] = 1$. (iv) $Pr_{SD_m}[p_{i+1} \mid p_i, a_i] = 1$.*

Using Proposition 3, probabilistic inferences can be performed. For example,

1. Given that P is false at time 0, what is the probability that P remains false at time 1?
   (answer: $Pr_{SD_1}[\neg p_1 \mid \neg p_0] = 1 - 0.5\lambda$)
2. Given that P is false at time 1, what is the probability that P was false at time 0?
   (answer: $Pr_{SD_1}[\neg p_0 \mid \neg p_1] = 1$)

Under the MLN semantics, this program specifies a different probabilistic transition system (shown in Figure 2), which is different from the commonsense understanding of the domain. For example, for Question 2, the MLN semantics gives $Pr[\neg p_0 \mid \neg p_1] = 0.5$, which means that even when

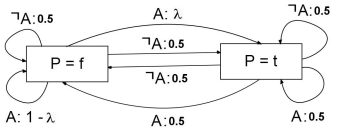

Figure 2: The transition system described by the MLN semantics

P is false, there is a high chance that P was true at the previous step, although there is no action that can cause P to be false. This is because under the MLN semantics, the rule "$P_{i+1} \leftarrow A_i, Aux_i$" does not express causal relations.

Note that the use of Aux_i is similar to the idea of probabilistic atoms in ProbLog. However, ProbLog does not accept rules such as (8)–(10).

Probabilistic Answer Set Planning: Wolf, Sheep, Cabbage Puzzle

McCarthy presented about 20 elaborations of the Missionaries and Cannibals puzzle to illustrate the concept of elaboration tolerance (McCarthy 1998). Most of these challenges involve logical reasoning only, but Elaboration #12 involves probabilistic reasoning as well, in which a boat can be stolen with the probability 1/10 when a cannibal is alone in the boat. The original domain can be represented in LP$^{\mathrm{MLN}}$, by taking advantages of features carried over from ASP. Further, with the probabilistic reasoning capability added to ASP, this elaboration can be handled in LP$^{\mathrm{MLN}}$.

For simplicity, we consider a variant of the Wolf, Sheep and Cabbage puzzle, where the objects left by themselves without the farmer can be eaten in accordance with the food chain. The domain can be formalized as follows. Let o, o_1, o_2 be schematic variables that range over $\{\mathit{Wolf}, \mathit{Sheep}, \mathit{Cabbage}\}$, and let l, l_1, l_2 be schematic variables that range over $\{L_1, L_2\}$. The states are described by the following rules:

$$\begin{array}{ll} \alpha: & \bot \leftarrow \mathit{OnBoat}_i(o_1), \mathit{OnBoat}_i(o_2)\ \ (o_1 \neq o_2) \\ \alpha: & \mathit{Loc}_i(o,l) \leftarrow \mathit{OnBoat}_i(o), \mathit{LocBoat}_i(l). \end{array}$$

The effects of actions are represented by the following LP$^{\mathrm{MLN}}$ rules.

$$\begin{array}{lrcl} \alpha: & \mathit{OnBoat}_{i+1}(o) & \leftarrow & \mathit{GetOnBoat}_i(o) \\ \alpha: & {\sim}\mathit{OnBoat}_{i+1}(o) & \leftarrow & \mathit{GetOffBoat}_i(o) \\ \alpha: & \mathit{LocBoat}_{i+1}(l) & \leftarrow & \mathit{MoveBoat}_i(l). \end{array}$$

The commonsense law of inertia for each fluent is specified by the following LP$^{\mathrm{MLN}}$ rules.

$$\begin{array}{lrcl} \alpha: & \{\mathit{Loc}_{i+1}(o,l)\}^{\mathrm{ch}} & \leftarrow & \mathit{Loc}_i(o,l) \\ \alpha: & \{\mathit{LocBoat}_{i+1}(l)\}^{\mathrm{ch}} & \leftarrow & \mathit{LocBoat}_i(l) \\ \alpha: & \{\mathit{OnBoat}_{i+1}(o)\}^{\mathrm{ch}} & \leftarrow & \mathit{OnBoat}_i(o) \\ \alpha: & \{{\sim}\mathit{OnBoat}_{i+1}(o)\}^{\mathrm{ch}} & \leftarrow & {\sim}\mathit{OnBoat}_i(o). \end{array}$$

Now we consider an elaboration in which with probability p, the wolf does not eat the sheep, and with probability q, the sheep does not eat the cabbage even when the farmer is not present. To handle this elaboration we introduce auxiliary atoms P_i and Q_i for each step i, and specify the probabilities as follows.

$$\begin{array}{lll} ln(p): & P_i & \\ ln(1-p): & \leftarrow P_i & \end{array} \qquad \begin{array}{lll} ln(q): & Q_i & \\ ln(1-q): & \leftarrow Q_i. & \end{array}$$

The success of a plan is defined by

$$\begin{array}{ll} \alpha: & \mathit{SheepEaten} \leftarrow \mathit{Loc}_i(\mathit{Wolf},l), \mathit{Loc}_i(\mathit{Sheep},l), \\ & \qquad \mathit{not\ LocBoat}_i(l), \mathit{not}\ P_i \\ \alpha: & \mathit{CabbageEaten} \leftarrow \mathit{Loc}_i(\mathit{Sheep},l), \mathit{Loc}_i(\mathit{Cabbage},l), \\ & \qquad \mathit{not\ LocBoat}_i(l), \mathit{not}\ Q_i \\ \alpha: & \mathit{Success} \leftarrow \mathit{Loc}_{\mathrm{maxstep}}(\mathit{Wolf},L_2), \\ & \quad \mathit{Loc}_{\mathrm{maxstep}}(\mathit{Sheep},L_2), \mathit{Loc}_{\mathrm{maxstep}}(\mathit{Cabbage},L_2), \\ & \quad \mathit{not\ SheepEaten}, \mathit{not\ CabbageEaten}. \end{array}$$

In addition to these rules, we also need rules that specify executability of actions, rules that define the uniqueness and existence of multi-valued fluents, rules that specify all actions as exogenous, and rules that define the initial states. Due to lack of space, we skip these rules.

While the minimal length plan for the original puzzle involves 17 actions of loading, moving and unloading, the elaboration has 6 new minimal length plans involving 11 actions only, two of which with $p \times p$ probability of success, two with $q \times q$, and two with $p \times p \times q \times q$. Since the program is tight, we could check the result using Alchemy by the method we introduced earlier.

Related Work

LP$^{\mathrm{MLN}}$ is related to many earlier work. Only some of them are mentioned here due to lack of space. We have already shown that ProbLog can be viewed as a special case of LP$^{\mathrm{MLN}}$. Other probabilistic logic programming languages that are based on distribution semantics are PRISM (Sato and Kameya 1997), ICL (Poole 1997) and LPADs (Vennekens et al. 2004). LP$^{\mathrm{MLN}}$ can embed a language with the distribution semantics, but in general it does not distinguish between probabilistic atoms and derived atoms, and the weights can be associated with arbitrary rules.

The logical component of LP$^{\mathrm{MLN}}$ is the expressive stable model semantics. In this sense, P-Log (Baral, Gelfond, and Rushton 2009), another approach to extending the language of ASP to handle probabilistic reasoning, is closely related. The LP$^{\mathrm{MLN}}$ formalization of probabilistic transition systems is related to P$\mathcal{C}$+ (Eiter and Lukasiewicz 2003), which extends $\mathcal{C}$+ for probabilistic reasoning about actions. In comparison with other approaches to formalizing probabilistic transitions, such as (Kersting, De Raedt, and Raiko 2006), the LP$^{\mathrm{MLN}}$ formalization is more elaboration tolerant and rich in knowledge representation due to the way that LP$^{\mathrm{MLN}}$ combines nonmonotonic reasoning and probabilistic reasoning in a single framework.

Conclusion

We introduced the semantics of LP$^{\mathrm{MLN}}$, which combines ASP and MLN in a single framework. The work presented here calls for more future work. One may design a native computation algorithm for turning LP$^{\mathrm{MLN}}$ to weighted model counting, which would be feasible to handle certain non-tight programs. We expect many results established in answer set programming may carry over to Markov Logic Networks, and vice versa, which may provide a new opportunity for probabilistic answer set programming.

Acknowledgements We are grateful to 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 Grant IIS-1319794, South Korea IT R&D program MKE/KIAT 2010-TD-300404-001, and ICT R&D program of MSIP/IITP 10044494 (WiseKB).

References

Babb, J., and Lee, J. 2015. Action language $\mathcal{BC}$+: Preliminary report. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. To appear.

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

Eiter, T., and Lukasiewicz, T. 2003. Probabilistic reasoning about actions in nonmonotonic causal theories. In *Proceedings Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, 192–199. Morgan Kaufmann Publishers.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2006. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence* 47:79–101.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2013. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming* 1–44.

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.

Kersting, K.; De Raedt, L.; and Raiko, T. 2006. Logical hidden markov models. *J. Artif. Intell. Res.(JAIR)* 25:425–456.

Lee, J.; Lifschitz, V.; and Palla, R. 2008. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 472–479.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 503–508. Professional Book Center.

Lifschitz, V., and Razborov, A. 2006. Why are there so many loop formulas? *ACM Transactions on Computational Logic* 7:261–268.

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

McCarthy, J. 1998. Elaboration tolerance[6]. In *Working Papers of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning*.

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

Raedt, L. D.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2462–2467.

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

Sato, T., and Kameya, Y. 1997. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, 1330–1335.

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.

[6] `http://www-formal.stanford.edu/jmc/elaboration.html`