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 concept of Markov Logic Networks. Each stable model is associated with the probability that is obtained from the weights of the rules from which the stable model is obtained. The proposed language, called LP$^{\text{MLN}}$, combines the advantages of the stable model semantics and Markov Logic Networks. We demonstrate the potentials of LP$^{\text{MLN}}$ by formalizing a few commonsense reasoning problems that require both logical and probabilistic reasoning.

Introduction

Logic programs under the stable model semantics 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 logic approaches, ASP is not well suited for handling uncertainty.

Markov Logic Networks (MLN) is a successful approach to combining first-order logic and a probabilistic graphical model in a single representation. There, each formula is associated with weights, 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 several implementations of MLN. However, MLN does not correctly represent inductive definitions, such as transitive closure, as it is based on first-order logic semantics.

We introduce a simple approach to combining the two successful formalisms—logic programs under the stable model semantics and Markov Logic Networks. In fact, the proposed language, which we call LP$^{\text{MLN}}$, is a proper generalization of each of them: an ASP program is a special case of an LP$^{\text{MLN}}$ program in which all rules have the so-called hard weight. MLN can be easily embedded in LP$^{\text{MLN}}$, and the other direction of embedding is also possible via the concept of loop formulas, which is similar to the reduction of logic programs to classical propositional logic known in answer set programming. This reduction allows us to use an implementation of MLN to compute LP$^{\text{MLN}}$ under certain conditions, similar to the way ASP programs can be computed by SAT solvers.

The paper is organized as follows. We first review the stable model semantics and Markov Logic Networks. Then we show how the two formalisms can be merged together resulting in LP$^{\text{MLN}}$. After discussing how it is related to logic programs and MLN, we demonstrate the potentials of LP$^{\text{MLN}}$ by representing a few commonsense reasoning problems that require both logical and probabilisitic reasoning.

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

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

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

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

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 "$A_1 \vee \dots \vee A_k \leftarrow A_{k+1} \wedge \dots \wedge A_m$" for all rules (2) in Π such that $I \models \neg A_{m+1} \wedge \dots \wedge \neg A_n \wedge \neg\neg A_{n+1} \wedge \dots \wedge \neg\neg A_p$. 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 the reduct of Π relative to I.[2] For example, for

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

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

the program

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

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." Formulas with the hard weight are called "hard formulas." 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 by

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

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

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 formulas that it satisfies. For every interpretation (i.e., possible world) I, there is a maximal subset of formulas in the MLN that I satisfies. This subset is $\mathbb{L}_I$, and the weight of I is related to the sum of 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). Each such ground formula is associated with the same weight as that of the formula in the non-ground MLN from which it is obtained.

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

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

An $\mathrm{LP}^{\mathrm{MLN}}$ 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 $\mathrm{LP}^{\mathrm{MLN}}$ program is *ground* if its rules contain no variables. Any $\mathrm{LP}^{\mathrm{MLN}}$ program $\mathbb{P}$ of signature σ can be turned into a ground $\mathrm{LP}^{\mathrm{MLN}}$ program $gr_\sigma[\mathbb{P}]$, whose rules are obtained from the rules of Π by replacing every variable with every ground terms of σ. Each such ground rule is associated with the same weight as that of the rule in $\mathbb{P}$ from which it is obtained. Any $\mathrm{LP}^{\mathrm{MLN}}$ program $\mathbb{P}$ of σ can be identified with its ground instance $gr_\sigma[\mathbb{P}]$.

Thus the syntax of $\mathrm{LP}^{\mathrm{MLN}}$ defines a set of weighted rules, which can be viewed as a special case of the syntax of MLN by identifying rules with implications.

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

For any ground $\mathrm{LP}^{\mathrm{MLN}}$ 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 *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 Π of the rules in $\mathbb{P}$ for which I is a stable model (under the standard stable model semantics). Unlike MLN, such a subset may not necessarily exist, in which case, the weight of the interpretation is assumed to be 0. However, if such one exists, there is a unique maximal subset for which I is a stable model, and this set turns out to be exactly $\mathbb{P}_I$. This interesting fact follows from the proposition below.

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.*

In other words, 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$^{\text{MLN}}$ *program* $\mathbb{P}$ *to be*

$$\begin{array}{lll} 1: & p & (r_1) \\ 2: & q & (r_2) \\ \alpha: & r \leftarrow p, q. & (r_3) \end{array}$$

$\Pi_{\mathbb{P}}$ *is the non-weighted program obtained from* $\mathbb{P}$*, and has only one stable model* $\{p, q, r\}$*, while* $\mathbb{P}$ *has* 4 *stable models with associated probabilities, among which* $\{p, q, r\}$ *has the highest probability. The following table shows the probability distribution over possible worlds under the* LP$^{\text{MLN}}$ *semantics as well as under the MLN semantics (where rules are identified with implications).* Z_1 *is* $1 + e + e^2 + e^3$ *and* Z_2 *is* $2 + 2e + 2e^2 + e^3$*.*

I	$\mathbb{P}_I$	$W_{\mathbb{P}}(I)$ *in* LP$^{\text{MLN}}$	$Pr_{\mathbb{P}}[I]$ *in* LP$^{\text{MLN}}$	$Pr_{\mathbb{P}}[I]$ *in MLN*
$\emptyset$	$\{r_3\}$	e^{α}	$1/Z_1$	$1/Z_2$
$\{p\}$	$\{r_1, r_3\}$	$e^{1+\alpha}$	e/Z_1	e/Z_2
$\{q\}$	$\{r_2, r_3\}$	$e^{2+\alpha}$	e^2/Z_1	e^2/Z_2
$\{r\}$	$\{r_3\}$	0	0	$1/Z_2$
$\{p, q\}$	$\{r_1, r_2\}$	e^3	0	0
$\{q, r\}$	$\{r_2, r_3\}$	0	0	e^2/Z_2
$\{p, r\}$	$\{r_1, r_3\}$	0	0	e/Z_2
$\{p, q, r\}$	$\{r_1, r_2, r_3\}$	$e^{3+\alpha}$	e^3/Z_1	e^3/Z_2

Example 2 *Consider an* LP$^{\text{MLN}}$ *program* $\mathbb{P}$ *to be*

$$\begin{array}{llll} 1: & p \leftarrow q & (r_1) & \qquad 2: \ p \leftarrow not\ r \quad (r_3) \\ 1: & q \leftarrow p & (r_2) & \qquad 3: \ r \leftarrow not\ p \quad (r_4) \end{array}$$

$\Pi_{\mathbb{P}}$ *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	$\mathbb{P}_I$	$Pr_{\mathbb{P}}[I]$ *in* LP$^{\text{MLN}}$	$Pr_{\mathbb{P}}[I]$ *in MLN*
$\emptyset$	$\{r_1, r_2\}$	e^2/Z_1	e^2/Z_2
$\{p\}$	$\{r_1, r_3, r_4\}$	e^6/Z_1	e^6/Z_2
$\{q\}$	$\{r_2\}$	0	e^1/Z_2
$\{r\}$	$\{r_1, r_2, r_3, r_4\}$	e^7/Z_1	e^7/Z_2
$\{p, q\}$	$\{r_1, r_2, r_3, r_4\}$	e^7/Z_1	e^7/Z_2
$\{q, r\}$	$\{r_2, r_3, r_4\}$	0	e^6/Z_2
$\{p, r\}$	$\{r_1, r_3, r_4\}$	0	e^6/Z_2
$\{p, q, r\}$	$\{r_1, r_2, r_3, r_4\}$	0	e^7/Z_2

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$^{\text{MLN}}$ *program.*

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

In each bullet, the reverse direction does not hold as the examples above illustrate.

Example 3 *(Fierens* et al. *2013) notes that "Markov Logic has the drawback that it cannot express (non-ground) inductive definitions." This limitation is overcome in* LP$^{\text{MLN}}$ *as it adopts the stable model semantics in place of the standard FO semantics in MLN. For instance, the following* LP$^{\text{MLN}}$ *program describes the probabilities of paths, which are induced by the probabilities of the edges that participate in forming the paths.*

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

Relating LP$^{\text{MLN}}$ to ASP and MLN

Relation to Answer Set Programming

Any logic program under the stable model semantics can be turned into an LP$^{\text{MLN}}$ 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$^{\text{MLN}}$ 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 stable models of* LP$^{\text{MLN}}$ *program* $\mathbb{P}_\Pi$ *coincide, and all stable models of* $\mathbb{P}_\Pi$ *have the same probability.*

For example, consider the logic program $\Pi = \{p. \quad q. \quad r \leftarrow p, q\}$. $\mathbb{P}_\Pi$ is $\{\alpha : p. \quad \alpha : q. \quad \alpha : r \leftarrow p, q\}$, which has only one stable model $\{p, q, r\}$, whose probability is 1. While $\{p\}$ is a stable model of the program in Example 1, it is not a stable model of the above program: $Pr_{\mathbb{P}_\Pi}[I] = \lim_{\alpha\to\infty} \frac{e^{2\alpha}}{e^{3\alpha}} = \lim_{\alpha\to\infty} \frac{1}{e^{\alpha}} = 0$. In fact, any interpretation that does not satisfy at least one of the hard rules is not a stable model.

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

The idea of softening rules in LP$^{\text{MLN}}$ 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$^{\text{MLN}}$, and do not have a probabilistic semantics.

Embedding MLN in LP$^{\text{MLN}}$

MLN can be easily embedded into LP$^{\text{MLN}}$. More precisely, any MLN $\mathbb{L}$ whose formulas have the form (2) can be turned into an LP$^{\text{MLN}}$ program $\mathbb{P}_\mathbb{L}$ so that the models of $\mathbb{L}$ coincides with the stable models of $\mathbb{P}_\mathbb{L}$. LP$^{\text{MLN}}$ program $\mathbb{P}_\mathbb{L}$ is obtained from $\mathbb{L}$ by adding

$$w: \quad A \leftarrow not\ not\ A$$

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 turning 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 2, the models of $\mathbb{P}$ and the stable models of $\mathbb{P} \cup \{w : A \leftarrow \mathit{not\ not}\ A \mid A \in \{p, q, r\}\}$ coincide.

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 probability distributions can be computed using existing implementations of MLN.

Review: Loop Formulas for Logic Programs We write ground rule (1) in the form

$$A \leftarrow B, N \tag{4}$$

where A is $A_1; \ldots; A_k$, B is $A_{k+1}, \ldots, A_m$, and N is

$$\mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n, \mathit{not\ not}\ A_{n+1}, \ldots, \mathit{not\ not}\ A_p\ .$$

We identify A with the set $\{A_1, \ldots, A_k\}$, and B with the set $\{A_{k+1}, \ldots, A_m\}$.

The dependency graph of a ground (disjunctive) program is defined as follows: the vertices of the graph are the ground atoms in the signature, and its edges go from the elements of A to the elements of B for all rules (4) of the program.

A *loop* of a formula F is a nonempty set of ground atoms such that the subgraph of the dependency graph of F induced by that set is strongly connected.

For any set Y of ground atoms, the *external support formula* of Y, denoted by $ES_\Pi(Y)$, is the disjunction of the formulas

$$B^\wedge \wedge N \wedge \bigwedge_{a \in A \setminus Y} \neg a \tag{5}$$

for all rules (4) of Π such that $A \cap Y \neq \emptyset$ and $B \cap Y = \emptyset$. The loop formula of Y, denoted by $LF_\Pi(Y)$, is $Y^\wedge \rightarrow ES_\Pi(Y)$.[3]

Theorem 3 *(Ferraris* et al. *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

$$\begin{array}{lcllcl} p & \rightarrow & q \vee \neg r & \qquad r & \rightarrow & \neg p \\ q & \rightarrow & p & \qquad p \wedge q & \rightarrow & \neg r \end{array} \tag{6}$$

[3]For any set Y of atoms, $Y^\wedge$ denotes the conjunction of the elements of Y.

The stable models $\{p, q\}$, $\{r\}$ of (3) are exactly the models of (3) that satisfy (6).

From LP$^{\mathrm{MLN}}$ to MLN The following theorem tell us how to turn LP$^{\mathrm{MLN}}$ programs 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 $\mathbb{P}_\alpha \cup \{LF_{\Pi_\mathbb{P}}(L) \mid L \text{ is a loop of } \Pi_\mathbb{P}\}$ 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 4 *When $\mathbb{P}$ is the LP$^{\mathrm{MLN}}$ program in Example 2, $\mathbb{L}_\mathbb{P}$ is the union of $\mathbb{P}$ and the loop formulas* (6) *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 2.*

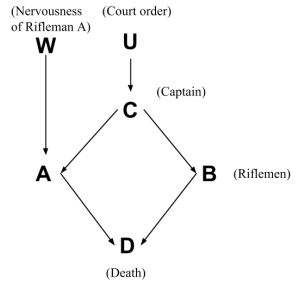
In principle, this theorem allows us to use an existing implementation of MLN, such as Alchemy and Tuffy, to compute LP$^{\mathrm{MLN}}$ programs. In the case when the program is *tight* (that is, its dependency graph is acyclic (Lee 2005)), the size of loop formulas is linear in the size of input programs. In the next section, we formalize a few examples in tight LP$^{\mathrm{MLN}}$ programs, and compute their stable models using Alchemy.

In general, it is known that the number of loop formulas blow up (Lifschitz and Razborov 2006), so adding all loop formulas at once is not a feasible idea. 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 3.

Further Examples

Firing Squad Example

The rule semantics is more appropriate than the standard FO semantics to express causality. The same advantage is carried over to LP$^{\mathrm{MLN}}$.



The Firing Squad example is from Section 7.1.2 of (Pearl 2000), which illustrates the concept of probabilistic causal models by Pearl. We show how this example can be represented in LP$^{\mathrm{MLN}}$ program FS. U denotes "The court orders the execution,", C denotes "The captain gives a signal," A denotes "Rifleman A shoots," B denotes "Rifleman B shoots," D denotes "The prisoner dies," and W denotes "Rifleman A is nervous." There is a probability p that the court has ordered the execution; rifleman A has a probability q of pulling the trigger out of nervousness. The left column represents the probabilities for exogenous variables U and W. The right column represents the causal rules for the endogenous variables.

$$
\begin{array}{rl}
ln(p) : & U \\
ln(1-p) : & \leftarrow U \\
ln(q) : & W \\
ln(1-q) : & \leftarrow W
\end{array}
\qquad
\begin{array}{rl}
\alpha : & C \leftarrow U \\
\alpha : & A \leftarrow C \\
\alpha : & A \leftarrow W \\
\alpha : & B \leftarrow C \\
\alpha : & D \leftarrow A \\
\alpha : & D \leftarrow B
\end{array}
$$

According to the semantics of LP^{MLN}, the stable models and their weights are shown in the following table.

Stable Models (I)	$W_{FS}(I)/e^{6\alpha}$
$I_1 = \emptyset$	$e^{ln(1-p)+ln(1-q)} = (1-p)(1-q)$
$I_2 = \{W, A, D\}$	$e^{ln(1-p)+ln(q)} = (1-p)q$
$I_3 = \{U, C, A, B, D\}$	$e^{ln(p)+ln(1-q)} = p(1-q)$
$I_4 = \{U, W, C, A, B, D\}$	$e^{ln(p)+ln(q)} = pq$

Using the method in the previous section, we can automate query answering for this domain using Alchemy. The following file `fs.mln` encodes the above LP^{MLN} description together with loop formulas. In the input language of Alchemy, hard rules are denoted by appending a period; soft rules are prepended by weights, and do not end with a period. We assume that $p = 0.6$, and $q = 0.1$.

```
% File 'fs.mln'
// Declarations          // Rules
U                        U => C.
W                        C => A.
C                        W => A.
A                        C => B.
B                        A => D.
D                        B => D.

-0.5108 U                // Loop Formulas
-0.9163 !U               C => U.
-2.3026 W                A => C v W.
-0.1054 !W               B => C.
                         D => A v B.
```

The command line for query answering in Alchemy is: `infer -i fs.mln -r out -e fs.db -q "U; A; B; C; D"` In each query below we put different evidences in `fs.db`.

1. (*Prediction*) Given that the court has not ordered the execution, what is the probability that the prisoner is dead?

   This query asks to compute $Pr_{FS}[d \mid \neg u]$,[4] which is the same as $\frac{W_{FS}(I_2)}{W_{FS}(I_1)+W_{FS}(I_2)} = q$. In Alchemy, we put the evidence `!U` into `fs.db`.

2. (*Abduction*) Given that the prisoner is dead, what is the probability that the court has ordered the execution?

   This query asks to compute $Pr_{FS}[u \mid d]$, which is the same as $\frac{W_{FS}(I_2)+W_{FS}(I_3)}{W_{FS}(I_2)+W_{FS}(I_3)+W_{FS}(I_4)} = \frac{p}{1-(1-p)(1-q)}$. In Alchemy, this query can be automated by putting `D` into `fs.db`.

3. (*Transduction*) Given that rifleman A shot, what is the probability that rifleman B shot as well?

   This query asks to compute $Pr_{FS}[b \mid a]$, which happens to be the same as the answer to the second query. In Alchemy, this query can be automated by putting `A` into `fs.db`.

This method can be extended to counterfactual queries, such as "Given that the prisoner is dead, what is the probability that the prisoner were not dead if rifleman A had not shot?" The encoding uses two copies of worlds and is similar to the one in (Baral and Hunsaker 2007), which formalizes the example in P-log. Due to lack of space we will discuss this query in a long version of the paper.

Representing Probabilistic Transitions

One of the most successful applications of ASP is in 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 ASP representations of transition systems with weights, the LP^{MLN} semantics gives an intuitive encoding of the probabilistic transitions in MDP.

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^{MLN} representation is as follows. We write $\{H\} \leftarrow Body$ to denote the rule $H \leftarrow Body, not\ not\ H$. This expression is called a "choice rule" in ASP.

$$
\begin{array}{rl}
ln(\lambda) : & Aux_i \\
ln(1-\lambda) : & \leftarrow Aux_i \\
\alpha : & \leftarrow P_i, {\sim}P_i \\
\alpha : & \leftarrow not\ P_i, not\ \ {\sim}P_i \\
\alpha : & P_{i+1} \leftarrow A_i, Aux_i
\end{array}
\qquad
\begin{array}{rl}
\alpha : & \{P_{i+1}\} \leftarrow P_i \\
\alpha : & \{{\sim}P_{i+1}\} \leftarrow {\sim}P_i \\
\alpha : & \{A_i\} \\
\alpha : & \{P_0\} \\
\alpha : & \{{\sim}P_0\}
\end{array}
$$

Here 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 $i+1$. We denote this program, parameterized with m, by SD_m.

The first two rules of SD_m describes probabilistic choices for the success of action A_i. The 3rd and 4th rules represent that P_i and ${\sim}P_i$ are complementary. The 5th rule describes the probabilistic effect of an action A_i. The 6th and 7th rules describe the commonsense law of inertia using choice rules. The last three rules represent that action A can be executed arbitrarily, and the initial values of P and ${\sim}P$ are arbitrary.

It can be checked that the stable models of SD_0 correspond to the states of the transition system (vertices); the stable models of SD_1 correspond to the transitions (edges), and their probability corresponds to the probability of the transition.

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,*

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

[4] We use lower case d to denote that $D = \mathbf{t}$. Similar with other variables.

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

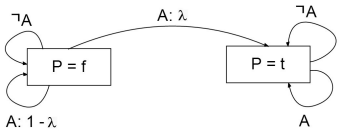


Figure 1: A transition system with probabilistic effects

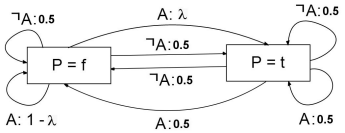


Figure 2: The transition system described by the MLN semantics

- *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 on this dynamic domain, 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 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 causality.

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 2007). 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^{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^{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 $\{Wolf, Sheep, Cabbage\}$, 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:

$$\begin{array}{ll} \alpha: & \bot \leftarrow OnBoat_i(o_1), OnBoat_i(o_2) \ \ (o_1 \neq o_2) \\ \alpha: & Loc_i(o,l) \leftarrow OnBoat_i(o), LocBoat_i(l). \end{array}$$

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

$$\begin{array}{ll} \alpha: & OnBoat_{i+1}(o) \leftarrow GetOnBoat_i(o) \\ \alpha: & {\sim}OnBoat_{i+1}(o) \leftarrow GetOffBoat_i(o) \\ \alpha: & LocBoat_{i+1}(l) \leftarrow MoveBoat_i(l) \end{array}$$

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

$$\begin{array}{ll} \alpha: & \{Loc_{i+1}(o,l)\} \leftarrow Loc_i(o,l). \\ \alpha: & \{LocBoat_{i+1}(l)\} \leftarrow LocBoat_i(l). \\ \alpha: & \{OnBoat_{i+1}(o)\} \leftarrow OnBoat_i(o). \\ \alpha: & \{{\sim}OnBoat_{i+1}(o)\} \leftarrow {\sim}OnBoat_i(o). \end{array}$$

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.

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

and define

$$\begin{array}{ll} \alpha: & SheepEaten \leftarrow Loc_i(Wolf,l), Loc_i(Sheep,l), \\ & \qquad not\ LocBoat_i(l), not\ P_i \\ \alpha: & CabbageEaten \leftarrow Loc_i(Sheep,l), Loc_i(Cabbage,l), \\ & \qquad not\ LocBoat_i(l), not\ Q_i. \end{array}$$

The success of a plan is defined by

$$\begin{array}{ll} \alpha: & Success \leftarrow Loc_m(Wolf,L_2), \\ & \qquad Loc_m(Sheep,L_2), Loc_m(Cabbage,L_2), \\ & \qquad not\ SheepEaten, not\ CabbageEaten. \end{array}$$

In addition to these rules, we also need rules that specify executability of actions, rules that defines the uniqueness and existence of multi-valued fluents, rules that specify all actions as exogenous, and rules that define the initial states. Due to the 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 $p \times p \times q \times q$.

Other Related Work and Conclusion

LP^{MLN} is related to many earlier work. Only a few of them are mentioned here due to lack of space. It is not difficult to embed ProbLog (Fierens *et al.* 2013) in LP^{MLN}. LP^{MLN} is also closely related to (Baral *et al.* 2009). The LP^{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 this paper, we introduced a variant of MLN, which allows rules under the stable model semantics instead of formulas in first-order logic. The idea is simple, but interestingly LP^{MLN} is quite general, and takes the advantages of both formalisms in a single framework.

The work presented here calls for more future work. One may design a native computation algorithm for LP^{MLN} 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.

References

Chitta Baral and Matt Hunsaker. Using the probabilistic logic programming language p-log for causal and counterfactual reasoning and non-naive conditioning. In *IJCAI*, pages 243–249, 2007.

Chitta Baral, Michael Gelfond, and J. Nelson Rushton. Probabilistic reasoning with answer sets. *TPLP*, 9(1):57–144, 2009.

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

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, 47:79–101, 2006.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, pages 1–44, 2013.

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

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

John McCarthy. Elaboration tolerance[6]. In progress, 2007.

Judea Pearl. *Causality: models, reasoning and inference*, volume 29. Cambridge Univ Press, 2000.

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

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