Handling Probability and Inconsistency in Answer Set Programming

Yi Wang

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, USA
(*e-mail:* {ywang485}@asu.edu)

submitted 29 April 2015; accepted 5 June 2015

Abstract

Answer Set Programming (ASP) is a powerful declarative computing paradigm that is especially suitable for modeling commonsense reasoning problems. However, the crisp nature of the underlying semantics, the stable model semantics, makes it difficult to handle reasoning domains involving probability and inconsistency. To address this issue, we present an extension of logic programs under the stable model semantics, where rules are associated with weights. Under our semantics, probabilistic commonsense domains where inconsistency might be involved can be represented in an intuitive and elaboration tolerant way. Our semantics extends MLN and logic programming under stable model semantics. We have shown that probabilistic action domains and Pearl's probabilisitic causal models can be represented, and various existing probabilistic logic programming frameworks can be embedded in our language. Future work includes further investigating the property of this language, devising algorithms for inference and learning in our language, and exploring various possible extensions of our language.

1 Introduction

Answer Set Programming (ASP) is a powerful logic programming paradigm that can model various commonsense reasoning problems elegantly and efficiently, thanks to useful constructs and efficient solvers developed. However, there are still many common reasoning problems that ASP cannot handle. A significant weakness of ASP that limits its applicability to real-world domains comes from the crisp nature of the underlying semantics, the stable model semantics. On one hand, statements involving probability cannot be expressed, on the other hand, no useful information can be derived when the knowledge based is inconsistent. In real-world applications where the knowledge base is often noisy, the capability of expressing probabilistic information is essential in specifying certainty degrees, and since information usually comes from many different sources which are not necessarily reliable, consistency should not be assumed. To see the problem, consider the following two situations:

Example 1
Consider the wolf, sheep, cabbage puzzle, where the objects left by themselves without the farmer can be eaten in accordance with the food chain. It is known that the puzzle can be modeled in ASP. Suppose we have an elaboration which says there is a chance that the wolf does not eat the sheep, or the sheep does not eat the cabbage, even when the farmer is

absent. We want to make the elaboration based on the formalization of the original problem description without dramatically changing the formalization.

Example 2
Suppose we have a knowledge base in ASP that says migratory birds and resident birds are two disjoint sets, and normally, every migratory bird or resident bird can fly. Properties on specific entities come from different data sources which are not necessary consistent with each other. There can be one data source saying Jo is a migratory bird and another saying Jo is a resident bird. In this case, we still want to conclude that Jo can fly.

In each of the two examples, the nonmonotonicity of ASP plays an essential role in modeling the problem. In Example 1, it is important in expressing the commonsense law of inertia, and in Example 2, it is important in expressing a defeasible rule. However, ASP cannot handle the inconsistency that can appear in Example 2, and probabilistic state definitions in Example 1.

To address the first aspect of the issue, i.e., handling probabilistic information in logic programming, various approaches to combining probability and logic programming have been proposed over the past decades, such as Poole's ICL (Poole 1997), CP-Logic (Vennekens et al. 2009), LPADs (Vennekens et al. 2004), ProbLog (Fierens et al. 2013). These frameworks allow a set of exogenous variables to be associated with probability, thus address the first aspect of the issue mentioned above. However, the expressivity of these languages is limited by their underlying logic programming semantics. For example, ProbLog is defined based on well-founded model semantics, which is not able to express generate-test style plan generation as allowed in ASP. Moreover, these languages are only defined for the case where the knowledge base is consistent, and thus do not address the second aspect of the issue mentioned above.

On the other hand, Markov Logic Networks (MLN) is a prominent approach in statistical relational learning. A Markov Logic Network is a set of weighted formulas, representing various pieces of knowledge of different confidence levels, based on which the subjective belief of each possible world can be determined. The weight-based semantics naturally handles probabilistic information and inconsistencies in logic knowledge base. However, since MLN is based on standard first-order semantics, it is weak for commonsense reasoning. For example, since transitive closure cannot be efficiently represented in MLN, inductive definitions such as "a friend of a friend is also a friend' can hardly be expressed.

My research focuses on an extension of logic programs under the stable model semantics, called LP^{MLN}(Lee and Wang 2015). The language combines MLN and ASP in a single framework without sacrificing any of their expressivity. Various probabilistic logic programming frameworks can be embedded in LP^{MLN}. Moreover, the weight-based semantics of LP^{MLN} handles inconsistency in an elegant way.

The goal of the research consists of the following: Firstly, we want to investigate the properties of LP^{MLN}. It is worth considering whether many known results from classical ASP can be carried over to this new framework, such as strong equivalence and splitting theorem. We also would like to have formal comparisons with other related works and in this way examine the applicability of the framework theoretically. Secondly, we want to design efficient algorithms for inference and learning under our semantics to have an

implementation. This will help us further understand this semantics from a practical perspective and produce an experiment platform to facilitate applications of this framework. Finally, we would like to consider possible extensions of the language, such as introducing advanced constructs in ASP to this probabilistic setting and defining a probabilistic action language based on the semantics. Based on this work, we hope to achieve a single framework where machine learning and knowledge representation seamlessly work together. Knowledge provided by humans and machine learnt rules are tightly combined to facilitate commonsense reasoning over domains involving uncertainty.

This paper will give a summary of my research, including the background (Section 2), preliminary results accomplished (Section 3), related work (Section 4) and the current status of the research (Section 5).

2 Background

2.1 Stable Model Semantics

We review the stable model semantics in this section. For simplicity, we only discuss the special case where each formula is of the rule form.

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

A *rule* over σ is of the form

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

($0 \leq k \leq m \leq n \leq p$) where all A_i are atoms of σ possibly containing object variables. We write $\{A_1\}^{\mathrm{ch}} \leftarrow \mathit{Body}$ to denote the rule $A_1 \leftarrow \mathit{Body}, \mathit{not\ not}\ A_1$. This expression is called a "choice rule" in ASP.

We will often identify (1) with the implication:

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

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 *(deterministic) stable model* of Π (denoted by $I \models_{\mathrm{SM}} \Pi$) if I is a minimal Herbrand model of Π^I. (Minimality is in terms of set inclusion. We identify an Herbrand interpretation with the set of atoms that are true in it.) For example, the stable models of the program

$$P \leftarrow Q \qquad Q \leftarrow P \qquad P \leftarrow \mathit{not}\ R \qquad R \leftarrow \mathit{not}\ P \tag{3}$$

are $\{P, Q\}$ and $\{R\}$. The reduct relative to $\{P, Q\}$ is $\{P \leftarrow Q.\ \ Q \leftarrow P.\ \ P.\}$, for which $\{P, Q\}$ is the minimal model; the reduct relative to $\{R\}$ is $\{P \leftarrow Q.\ \ Q \leftarrow P.\ \ R.\}$, 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.

The semantics is extended to allow some useful constructs, such as aggregates and abstract constraints (e.g., (Niemelä and Simons 2000; Faber et al. 2004; Ferraris 2005; Son et al. 2006; Pelov et al. 2007)), which is proved to be useful in many KR domains.

2.2 Markov Logic Network

The following is a review of MLNs from (Richardson and Domingos 2006). 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 "infinite weight." We say that $\mathbb{L}$ 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 $Pr_\mathbb{L}[I] \neq 0$.

The basic idea of MLNs 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, which 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 the ground formula is obtained.

3 Preliminary results accomplished

In this section, we briefly review the results published in (Lee and Wang 2015) and (Lee et al. 2015).

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

The syntax of LP$^{\mathrm{MLN}}$ defines a set of weighted rules. More precisely, an LP$^{\mathrm{MLN}}$ program $\mathbb{P}$ 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 a symbol α denoting the "infinite weight." We call rule $w : R$ *soft* rule if w is a real number, and *hard* rule if w is α.

We say that an LP$^{\mathrm{MLN}}$ program is *ground* if its rules contain no variables. We identify any LP$^{\mathrm{MLN}}$ program $\mathbb{P}$ of signature σ with a ground LP$^{\mathrm{MLN}}$ 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 the ground rule is obtained.

By $\overline{\mathbb{P}}$ we denote the logic program obtained from $\mathbb{P}$ by dropping the weights, i.e., $\overline{\mathbb{P}} = \{R \mid w : R \in \mathbb{P}\}$. By $\overline{\mathbb{P}}_I$ we denote the set of rules in $\overline{\mathbb{P}}$ which are satisfied by I.

The semantics of LP$^{\mathrm{MLN}}$ is inspired by Markov Logic Networks ((Richardson and Domingos 2006)). In MLN, a model does not have to satisfy all formulas, but each model is associated with the weight that is contributed by the subset of the formulas that the model satisfies. Likewise, in LP$^{\mathrm{MLN}}$, a stable model does not have to be obtained from the whole program, but each stable model is associated with the weight that is contributed by the subset of the rules from which the stable model is obtained.

Thus we define the *weight* of an interpretation I w.r.t. $\mathbb{P}$, denoted $W_{\mathbb{P}}(I)$, as

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

Let SM$[\mathbb{P}]$ be the set $\{I \mid I \text{ is a stable model of } \overline{\mathbb{P}}_I\}$. Notice that SM$[\mathbb{P}]$ is never empty because it always contains the empty set. It is easy to check that the set $\emptyset$ always satisfies $\overline{P}_\emptyset$, and it is the smallest set that satisfies the reduct $(\overline{P}_\emptyset)^\emptyset$.

Using this notion of a weight, we define the *probability* of an interpretation I under $\mathbb{P}$, denoted $Pr_{\mathbb{P}}[I]$, as follows. For any interpretation I,

$$Pr_{\mathbb{P}}[I] = \begin{cases} \lim\limits_{\alpha \to \infty} \frac{W_{\mathbb{P}}(I)}{\sum_{J \in \mathrm{SM}[\mathbb{P}]} W_{\mathbb{P}}(J)} & \text{if } I \in \mathrm{SM}[\mathbb{P}]; \\ 0 & \text{otherwise.} \end{cases}$$

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 MLNs. For each interpretation I, we try to find a maximal subset (possibly empty) of $\overline{\mathbb{P}}$ 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 $\overline{\mathbb{P}}$, and the probability distribution is over these stable models.

For any proposition A, $Pr_{\mathbb{P}}[A]$ is defined as usual:

$$Pr_{\mathbb{P}}[A] = \sum_{I:\ I \models A} Pr_{\mathbb{P}}[I].$$

As an example, consider an LP$^{\mathrm{MLN}}$ program $\mathbb{P}$.

$1:\ P \leftarrow Q \quad (r_1)$ $\qquad$ $1:\ Q \leftarrow P \quad (r_2)$ $\qquad$ $2:\ P \leftarrow \mathit{not}\ R \quad (r_3)$ $\qquad$ $3:\ R \leftarrow \mathit{not}\ P. \quad (r_4)$

I	$\mathbb{P}_I$	$Pr_{\mathbb{P}}[I]$
$\emptyset$	$\{r_1, r_2\}$	e^2/Z
$\{P\}$	$\{r_1, r_3, r_4\}$	e^6/Z
$\{Q\}$	$\{r_2\}$	0
$\{R\}$	$\{r_1, r_2, r_3, r_4\}$	e^7/Z
$\{P, Q\}$	$\{r_1, r_2, r_3, r_4\}$	e^7/Z
$\{Q, R\}$	$\{r_2, r_3, r_4\}$	0
$\{P, R\}$	$\{r_1, r_3, r_4\}$	0
$\{P, Q, R\}$	$\{r_1, r_2, r_3, r_4\}$	0

The probability of each interpretation are shown in the following table, where Z is $e^2 + e^6 + 2e^7$.

The (deterministic) stable models $\{P, Q\}$ and $\{R\}$ of $\Pi_{\mathbb{P}}$ are the (probabilistic) stable models of $\mathbb{P}$ with the highest probability. In addition, $\mathbb{P}$ has two other (probabilistic) stable models, which do not satisfy some rules in $\Pi_{\mathbb{P}}$ and is thus less probable.

Representing Transitive Closure in LP$^{\mathrm{MLN}}$: The following example illustrates that unlike MLN, transitive closure can be represented in LP$^{\mathrm{MLN}}$. Consider that there is a chance x influences y if x is a friend to y.

$$\begin{array}{ll} \alpha: & \mathit{Friend}(A, B). \\ \alpha: & \mathit{Friend}(B, C). \\ w: & \mathit{Influences}(x, y) \leftarrow \mathit{Friend}(x, y) \\ \alpha: & \mathit{Influences}(x, y) \leftarrow \mathit{Influences}(x, z), \mathit{Influences}(z, y). \end{array}$$

Note that the third rule is a soft rule: a person does not always influence his/her friend.

Under the LP$^{\mathrm{MLN}}$ semantics, A has a high probability to influence B, and so does B to C. The third rule leads to that A influences C with some smaller probability that is related to the two probabilities.

Handling Inconsistency in LP$^{\mathrm{MLN}}$: Example 2 can be easily handled in LP$^{\mathrm{MLN}}$. The ASP knowledge base is simply represented using hard rules:

$$\begin{array}{ll} \alpha: & Fly(x) \leftarrow MigratoryBird(x), \mathit{not}\ ab(x). \\ \alpha: & Fly(x) \leftarrow ResidentBird(x), \mathit{not}\ ab(x). \\ \alpha: & \leftarrow ResidentBird(x), MigratoryBird(x). \end{array}$$

Data source 1 says Jo is migratory bird, which is added to the knowledge base:

$$\alpha: \ MigratoryBird(Jo).$$

Similarly, data source 2 says Jo is resident bird:

$$\alpha: \ ResidentBird(Jo).$$

It can be checked that under LP$^{\mathrm{MLN}}$ semantics, we have

- $Pr\left[Fly(Jo)\right] = 1$, and
- $Pr\left[MigratoryBird(Jo)\right] = Pr\left[ResidentBird(Jo)\right] = \frac{2}{3}$.

Representing Probabilistic Action Domain in LP$^{\mathrm{MLN}}$: Example 1 can be modeled in LP$^{\mathrm{MLN}}$ as well. In addition to the ASP rules that define the original problem, we introduce auxiliary atoms P_i and Q_i for each step i, and specify the probabilities as follows.

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

The success of a plan is defined by

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

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 $p \times p \times q \times q$.

3.2 Relation to ASP and MLNs

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

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

Embed MLNs in LP^{MLN}**:** Any MLN $\mathbb{L}$ whose formulas have the form (2) can be turned into an LP^{MLN} program $\mathbb{P}_\mathbb{L}$ so that the models of $\mathbb{L}$ coincide with the stable models of $\mathbb{P}_\mathbb{L}$, keeping the same probability distribution.

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

$$w: \quad A \leftarrow \textit{not not } A$$

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 turn that into the rule form. For instance, $P \vee Q \vee \neg R$ is turned into $P \vee Q \leftarrow R$.

Turning LP^{MLN} **programs into MLNs:** It is known that the stable models of a logic program coincide with the models of a logic program plus all its loop formulas (Ferraris et al. 2006). This allows us to compute the stable models using SAT solvers. The method can be extended to LP^{MLN} so that their stable models along with the probability distribution can be computed using existing implementations of MLNs, such as Alchemy [1] and Tuffy. [2]

We define $\mathbb{L}_\mathbb{P}$ to be the union of $\mathbb{P}$ and $\{\alpha : LF_{\Pi_\mathbb{P}}(L) \mid L \text{ is a loop of } \Pi_\mathbb{P}\}$.

[1] http://alchemy.cs.washington.edu
[2] http://i.stanford.edu/hazy/hazy/tuffy

Theorem 3
For any LP$^{\mathrm{MLN}}$ program $\mathbb{P}$ such that

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

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.

It is known that the number of loop formulas may blow up (Lifschitz and Razborov 2006). As LP$^{\mathrm{MLN}}$ is a generalization of answer set programs, this blow-up is unavoidable also in the context of LP$^{\mathrm{MLN}}$. This calls for a computational method such as the incremental addition of loop formulas as in ASSAT (Lin and Zhao 2004). Or, when the program is *tight* (that is, its dependency graph is acyclic), the size of loop formulas is linear in the size of input programs (Lee 2005).

3.3 Relation to ProbLog, CP-Logic and LPADs

We have shown 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 Sato (1995). The formal result can be found in (Lee and Wang 2015). Here we illustrate the idea by an example.

The following ProbLog program

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

corresponds to the LP$^{\mathrm{MLN}}$ program

$$\begin{array}{rl} ln(0.6) : & P \\ ln(0.4) : & \leftarrow P \\ ln(0.4) : & Q \\ ln(0.6) : & \leftarrow Q \\ \alpha : & R \leftarrow P \\ \alpha : & R \leftarrow Q. \end{array}$$

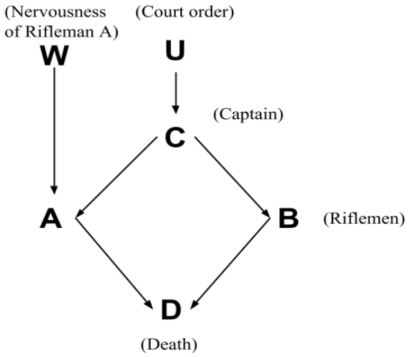
Syntactically, LP$^{\mathrm{MLN}}$ allows more general rules than ProbLog, such as disjunctions in the head, as well as the empty head and double negations in the body. Further, LP$^{\mathrm{MLN}}$ allows rules to be weighted as well as facts, and do not distinguish between probabilistic facts and derived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

It has been shown that Logic Programs with Annotated Disjunctions (LPADs) (Vennekens et al. 2004) and CP-logic (Vennekens et al. 2009) can be turned into ProbLog, and thus can be embedded in LP$^{\mathrm{MLN}}$ as well. It is worth noting that a main difference between ProbLog/ CP-Logic/LPADs and LP$^{\mathrm{MLN}}$ is that the former is based on the well-founded model semantics and the latter is on the stable model semantics. In the deterministic case, the similarity and the difference between them is well-known, and this carries over to the probabilistic setting.

3.4 Embedding Pearl's Probabilistic Causal Model

Pearl's Probabilistic causal model can be embedded in LP$^{\mathrm{MLN}}$. Again, here we only illustrate the idea by an example.

Consider the Firing Squad example discuss in (Pearl 2000, Sec 7.1.2).



$$C = U$$
$$A = C \vee W$$
$$B = C$$
$$D = A \vee B$$

$$P(U = \mathbf{t}) = p$$
$$P(W = \mathbf{t}) = q$$

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.

This example can represented in LP$^{\mathrm{MLN}}$ as the following program:

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

Counterfactual reasoning can also be handled in LP$^{\mathrm{MLN}}$, but it requires a more complicated translation that captures the twin network method (Balke and Pearl 1994).

4 Related Work

Our approach is closely related to P-Log (Baral et al. 2009), which is another approach to extending ASP to handle probabilistic reasoning. Probabilistic information can be expressed in P-log in quite an intuitive way altogether with classical ASP rules. However, P-log does not have an efficient implementation. Another problem with this framework is that it is mainly designed for human-written knowledge base, and it is not easy to incorporate machine learnt rules. Some other approaches include (Nickles and Mileo 2014), (Ng and Subrahmanian 1994), and (Saad and Pontelli 2005).

There are quite a few other approaches to extending logic programming to probabilistic settings. ProbLog (Fierens et al. 2013) is a well-known framework under this category. It combines a probabilistic semantics with the well-founded model semantics, making it possible to incorporate probabilistic information in a logic program. Poole's ICL (Poole 1997), CP-Logic (Vennekens et al. 2009) and LPADs (Vennekens et al. 2004) are also languages which allow probabilistic information to be represented through a logical rule based semantics. CP-logic is a probabilistic extension of FO(ID) (Denecker and Ternovska

2007). It is shown in (Vennekens et al. 2006), that CP-logic "almost completely coincides" with LPAD. It has been shown that CP-logic can be embedded in ProbLog. In (Vennekens et al. 2004), it is shown that Poole's ICL can be viewed as LPADs, and that acyclic LPAD program can be turned into ICL. As we mentioned in 3.3, these languages can be embedded in LP^{MLN}.

5 Current status of the research

In (Lee and Wang 2015), we have presented the semantics of this probabilistic extension of stable model and have shown that it properly generalizes ASP and MLNs. It is also shown in the paper that ProbLog can be embedded in our language and probabilistic action domains can be represented in our language.

After the publication of (Lee and Wang 2015), we further showed that Pearl's probabilistic causal model (Pearl 2000) can be embedded in our language. The relations between some other related works, such as (Poole 1997), (Vennekens et al. 2009) and (Vennekens et al. 2004), have also been discussed. These results are presented in the paper (Lee et al. 2015).

Currently we are working on formally characterizing LP^{MLN}'s capability of handling inconsistency, and developing a compiler for a simple but useful subset of programs ("tight" programs). As the next step, we will focus on designing native inference algorithms and establishing a learning framework.

Acknowledgements Sincere thanks to my mentor Joohyung Lee, and the anonymous reviewers for their useful comments. This work was partially supported by the National Science Foundation under Grant IIS-1319794.

References

BALKE, A. AND PEARL, J. 1994. Counterfactual probabilities: Computational methods, bounds and applications. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pp. 46–54. Morgan Kaufmann Publishers Inc.

BARAL, C., GELFOND, M., AND RUSHTON, J. N. 2009. Probabilistic reasoning with answer sets. *TPLP 9,* 1, 57–144.

DENECKER, M. AND TERNOVSKA, E. 2007. Inductive situation calculus. *Artificial Intelligence 171,* 5-6, 332–360.

FABER, W., LEONE, N., AND PFEIFER, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*.

FERRARIS, P. 2005. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 119–131.

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.

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

LEE, J., MENG, Y., AND WANG, Y. 2015. Markov logc style weighted rules under the stable model semantics. In *Technical Communications of the 31st International Conference on Logic Programming*. To appear.

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

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.

NG, R. AND SUBRAHMANIAN, V. 1994. Stable semantics for probabilistic deductive databases. *Information and computation 110,* 1, 42–83.

NICKLES, M. AND MILEO, A. 2014. Probabilistic inductive logic programming based on answer set programming.

NIEMELÄ, I. AND SIMONS, P. 2000. Extending the Smodels system with cardinality and weight constraints. In J. MINKER (Ed.), *Logic-Based Artificial Intelligence*, pp. 491–521. Kluwer.

PEARL, J. 2000. *Causality: models, reasoning and inference*, Volume 29. Cambridge Univ Press.

PELOV, N., DENECKER, M., AND BRUYNOOGHE, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *TPLP 7,* 3, 301–353.

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

RICHARDSON, M. AND DOMINGOS, P. 2006. Markov logic networks. *Machine Learning 62,* 1-2, 107–136.

SAAD, E. AND PONTELLI, E. 2005. Hybrid probabilistic logic programming with non-monotoic negation. In *In Twenty First International Conference on Logic Programming*.

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

SON, T. C., PONTELLI, E., AND TU, P. H. 2006. Answer sets for logic programs with arbitrary abstract constraint atoms. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence (AAAI)*.

VENNEKENS, J., DENECKER, M., AND BRUYNOOGHE, M. 2006. Representing causal information about a probabilistic process. In *Logics In Artificial Intelligence*, pp. 452–464.

VENNEKENS, J., DENECKER, M., AND BRUYNOOGHE, M. 2009. Cp-logic: A language of causal probabilistic events and its relation to logic programming. *TPLP 9,* 3, 245–308.

VENNEKENS, J., VERBAETEN, S., BRUYNOOGHE, M., AND A, C. 2004. Logic programs with annotated disjunctions. In *Proceedings of International Conference on Logic Programming (ICLP)*, pp. 431–445.