On the Relationship between P-log and LP^{MLN}*

Evgenii Balai and **Michael Gelfond**
Texas Tech University, Lubbock Texas
{evgenii.balai, michael.gelfond}@ttu.edu

Abstract

The paper investigates the relationship between knowledge representation languages P-log [Baral *et al.*, 2004] and LP^{MLN} [Lee *et al.*, 2015] designed for representing and reasoning with logic and probability. We give a translation from an important subset of LP^{MLN} to P-log which preserves probabilistic functions defined by LP^{MLN} programs and complements recent research [Lee and Wang, 2016] by the authors of LP^{MLN} where they give a similar translation from a subset of P-log to their language. This work sheds light on the different ways to treat inconsistency in both languages.

1 Introduction

Combining logic and probability has been one of the most important directions of artificial intelligence research in recent years. Many different languages and formalisms have been developed to represent and reason about both probabilistic and logical arguments, such as ProbLog [De Raedt *et al.*, 2007; Fierens *et al.*, 2015], PRISM [Sato, 1995; 2009], LPADs [Vennekens *et al.*, 2004], CP-Logic [Vennekens *et al.*, 2009], MLN [Richardson and Domingos, 2006], and others.

In this paper we focus on two such languages, P-log and LP^{MLN}. They are distinguished from other mentioned alternatives by their common logic base, Answer Set Prolog (ASP) [Gelfond and Lifschitz, 1991], a logical formalism modeling beliefs of a rational agent. ASP is powerful enough to naturally represent defaults, non-monotonically update the knowledge base with new information, define relations recursively, reason about causal effects of actions, etc. The language serves as the foundation of the so called Answer Set Programming paradigm [Marek and Truszczynski, 1999; Niemela, 1998] and has been used in a large number of applications [Brewka *et al.*, 2011].

An agent associated with an ASP knowledge base reasons about three degrees of belief – he can believe that p is $true$, believe that p is $false$, or remain uncommitted about his belief in p. In the latter case the truth of p remains $unknown$.

*This work was sponsored by NASA under the grant CertWare ABSA: Argument Based Safety Assurance, Grant #: NRA NNH11ZEA001N-SSAT2

An extension of ASP, called P-log, allows the reasoner to express and reason with finer, numerically expressed, gradation of the strength of his beliefs. In other words, it preserves the power of ASP and, in addition, allows an agent to do sophisticated probabilistic reasoning.

The main goal of the P-log designers was to provide the language and reasoning mechanism which can be used for clear and transparent modeling of knowledge involving logical and probabilistic arguments. There are a number of nontrivial scenarios formalized in [Baral *et al.*, 2009]. More computationally challenging scenarios, including probabilistic planning and diagnosis, can be found in [Zhu, 2012], where their P-log representations and performance analysis are given.

A new version of P-log , introduced in [Gelfond and Rushton, 2010], replaces ASP by its extension CR-Prolog [Balduccini and Gelfond, 2003], which expands logical power of ASP (and hence the original P-log) by allowing so called consistency restoring rules (cr-rules) used for restoring consistency of the program by a certain form of abductive reasoning.

Despite the presence of cr-rules, the underlying philosophy of P-log requires the corresponding knowledge base to be consistent. A rational reasoner is assumed to trust its rules and refuses to deal with a knowledge base containing statements p and $\neg p$. Possible means to ensure consistency of the program should be supplied by a knowledge engineer. This is natural from a theoretical goal of the authors but is also important in many practical applications, for example, where inconsistency of the knowledge base may be a sign of some errors in its design and therefore should be addressed by making the necessary changes.

The language LP^{MLN}, introduced in [Lee *et al.*, 2015], is based on a different philosophy. Its first goal seems to be similar to that of P-log – it is supposed to provide means for combining ASP based reasoning with reasoning about probability. But, in addition, the new language is aimed at providing a powerful (though somewhat less predictable) way of resolving inconsistencies which may appear in LP^{MLN} programs due to mechanical combination of different knowledge bases, designer mistakes, or some other reasons. The design of the language was influenced by Markov Logic Networks [Richardson and Domingos, 2006] and seems to be practically independent from P-log. As a result, the relationship between these two languages with seemingly similar goals

remains unclear. This paper is a step in remedying this situation. In particular, we give a translation from an important subset of LP$^{\text{MLN}}$ to P-log which preserves probabilistic functions defined by LP$^{\text{MLN}}$ programs. The work complements resent research[Lee and Wang, 2016] by Lee and Wang in which the authors give a translation from a subset of P-log to LP$^{\text{MLN}}$.

The rest of this paper is organized as follows. In section 2 we define the subset of P-log used in this paper. In section 3 we briefly describe the syntax and semantics of LP$^{\text{MLN}}$. In section 4 we describe a translation from LP$^{\text{MLN}}$ to P-log and define the correspondence between LP$^{\text{MLN}}$ programs and their P-log translations precisely. Section 5 uses the results from sections 4 to describe certain properties of probabilities defined by LP$^{\text{MLN}}$ programs. Section 6 concludes the paper by summarizing the obtained results and future work.

2 Language P-log

In this section we introduce a simplified version of P-log with consistency restoring rules from [Gelfond and Rushton, 2010] which is sufficient to define the translation from LP$^{\text{MLN}}$. We do so by considering a simple domain consisting of two adjacent rooms, r_1 and r_2, and a robot initially located in r_1. We present a P-log program, Π_0, modeling direct effects of the action $move$ of the robot attempting to enter the second room. The program is presented to illustrate the language constructs so we ignore concerns about its generality, elaboration tolerance, etc.

We start with declarations of objects and functions of the domain. In P-log such functions are usually referred to as *attributes*, while expressions of the form $f(\bar{x})$, where f is an attribute, are called *attribute terms*. We need the sort

$$step = \{0, 1\}$$

where 0 and 1 denote time-steps before and after the execution of the action respectively. We also need the sort

$$room = \{r_1, r_2\}$$

and attributes

$$move, broken : boolean$$

$$loc \ : \ step \rightarrow room$$

Here $move$ is true iff at step 0 the robot has attempted to move to room r_2; $broken$ holds if the robot has been broken and hence may exhibit some non-deterministic behavior; loc gives the location of the robot at a given step. Declarations of P-log are followed by the program rules. In our case we will have rules

$$loc(0) = r_1 \tag{1}$$

$$move \tag{2}$$

indicating that at step 0 the robot located in room r_1 attempted to move to room r_2. Here $move$ is a shorthand for $move = true$. We use this convention for all boolean functions: $f(\bar{x}) = true$ and $f(\bar{x}) = false$ are written as $f(\bar{x})$ and $\neg f(\bar{x})$ respectively.

As expected the effect of action $move$ for the well-functioning robot is given by the rule:

$$loc(1) = r_2 \leftarrow not\ broken, move. \tag{3}$$

If the robot is malfunctioning however we need to state that the effect of $move$ is random – the robot can still successfully move to room r_2 or to stay in room r_1. In P-log this is expressed by the following *random selection rule* r

$$[r]\ random(loc(1)) \leftarrow broken, move \tag{4}$$

which says that if the malfunctioning robot will attempt to move to room r_2 then, in the resulting state, attribute term $loc(1)$ will randomly take a value from the range of loc. The rules of the program described so far can be easily translated into regular ASP rules – we simply need to replace $random(loc(1))$ in the last rule by $(loc(1) = r_1\ or\ loc(1) = r_2)$, replace atoms of the form $f(\bar{x}) = y$ by $f(\bar{x}, y)$ and, for every term $f(\bar{x})$, add a constraint $\{\leftarrow\ f(\bar{x}) = y_1, f(\bar{x}) = y_2, y_1 \neq y_2\}$. In general, an atom of the form $random(f(\bar{x}))$ is replaced by the disjunction $f(\bar{x}) = y_1\ or\ \ldots\ or\ f(\bar{x}) = y_k$ where $\{y_1, \ldots, y_k\}$ is the range of f.

Answer sets of the translation of a P-log program Π into ASP are viewed as *possible worlds* of the probabilistic model defined by Π. It is easy to see that program Π^0 consisting of rules (1)–(4) has one such possible world $W_0 = \{move, loc(0) = r_1, loc(1) = r_2\}$[1] . Program $\Pi^1 = \Pi^0 \cup \{broken\}$ will have two possible worlds, $W_1 = \{broken, move, loc(0) = r_1, loc(1) = r_2\}$ and $W_2 = \{broken, move, loc(0) = r_1, loc(1) = r_1\}$. In the case of multiple possible worlds we need some device allowing to specify the numeric probabilities of possible values of random attributes. This is expressed in P-log through *causal probability statements*, or, simply, *pr-atoms*. A pr-atom takes the form

$$pr_r(f(\bar{x}) = y) = v$$

where $f(\bar{x})$ is a random attribute, y is a value from the range of f, and $v \in [0, 1]$ is the probability of y to be selected as the value of $f(\bar{x})$ as the result of firing random selection rule r. In case of Π^1 such pr-atoms may look as, say,

$$pr_r(loc(1) = r_1) = 0.3$$

and

$$pr_r(loc(1) = r_2) = 0.7$$

Unnormalized probabilistic measure of a possible world W is defined as the product of probabilities of the random atoms $f_1(\bar{x}_1) = y_1, \ldots, f_k(\bar{x}_k) = y_k$ from W. These probabilities are obtained from the corresponding pr-atoms. Normalized probabilistic measures and probability function on the sets of possible worlds and on the literals of the language are defined as usual. Let P_0 and P_1 be the probability functions defined by Π^0 and Π^1 respectively. W_0 has no random atoms, the empty product is 1, and hence the probabilistic measure of

[1] for convenience we will often identify original P-log literals with corresponding ASP ones (e.g, we will sometimes write $loc(1) = r_1$ in place of $loc(1, r_1)$)

W_0 and $P_0(loc(1) = r_1)$ are both equal to 1. The probabilistic measures of W_1 and W_2 are 0.7 and 0.3 respectively and hence $P_1(loc(1) = r_1) = 0.3$.

As mentioned in the introduction, a P-log program can be inconsistent. For instance, program $\Pi^2 = \Pi^0 \cup \{loc(1) = r_1\}$ has no possible worlds. To avoid this particular inconsistency the program designer can expand Π^0 by a cr-rule:

$$broken \stackrel{+}{\leftarrow} . \tag{5}$$

which allows to restore inconsistency of Π^2 by assuming that the robot is broken. Since the original program Π^0 is consistent, the resulting program, Π^0_{new} will define the same probabilistic model as Π^0. The program Π^2_{new}, consisting of Π^0_{new} and the fact $\{loc(1) = r_1\}$, unlike the program Π^2, will be consistent and have one possible world, W_2. The extension of Π^0 by a new information changed the probability of the robot being in room r_2 after execution of $move$ from 1 to 0.

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

In this section we give a brief summary of LP$^{\text{MLN}}$ ([Lee *et al.*, 2015]). We limit out attention to ground programs whose rules contain no double default negation $not\ not$ and no disjunction. To the best of our knowledge, no example in the literature demonstrating the use of LP$^{\text{MLN}}$ for formalization of knowledge uses these constructs. As usual, we may use rules with variables viewed as shorthands for the sets of their ground instances. A program of the language is a finite set of LP$^{\text{MLN}}$ *rules* – ground ASP rules preceded by a *weight*: symbol α or a real number. Rules of the first type are called *hard* while rules of the second are referred to as *soft*. Despite their name the hard rules are not really "hard". Their behavior is reminiscent of that of defaults. According to the semantics of the language the reasoner associated with a program constructs possible worlds with non-zero probability by trying to satisfy as many hard rules as possible. The satisfiability requirement for the soft rules and the use of their weights for assigning the probability measure to possible worlds of M are more subtle. In what follows we give the necessary definitions and illustrate them by an example of LP$^{\text{MLN}}$ program. Sometimes we abuse the notation and identify an LP$^{\text{MLN}}$ program M with its *ASP counterpart* obtained from M by dropping the weights of its rules. Stable models of such a counterpart will be referred to as *ASP models* of M. By M_I we denote the set of rules of M which are satised by an interpretation I of M. An interpretation W is a *possible world* of M if it is a ASP model of M_W. We will say that a possible world W *is supported by* M_W. As usual, by Ω_M we denote the set of all possible worlds of M. *Unnormalized measure* of a possible world $W \in \Omega_M$ (denoted by $w_M(W)$) is exp^{γ} where γ is the sum of weights of all rules of M satisfied by W. Note that, in case M contains rules with α-weights satisfied by W, $w_M(W)$ is not a numerical value and should be understood as a symbolic expression. The *probability function*, P_M, *defined by program* M is

$$P_M(W) = \lim_{\alpha \to \infty} \frac{w_M(W)}{\sum_{V \in \Omega_M} w_M(V)}$$

It is easy to check that P_M maps possible worlds of M into the interval $[0, 1]$ and satisfies standard axioms of probability.

As expected, *the probabilistic model* defined by M consists of Ω_M and P_M.

Let us now use LP$^{\text{MLN}}$ to formalize the stories from the previous section. Program M^0 will capture the first such story corresponding to P-log program Π^0. It clearly should contain rules (1) – (3) of Π^0. In addition, for every attribute f it must include a constraint

$$\leftarrow f(X) = Y_1, f(X) = Y_2, Y_1 \neq Y_2 \tag{6}$$

which is hidden in P-log semantics of Π^0. All these rules, however, should be supplied with some weights. Since we strongly believe that the rules are correct, we would like to preserve as many of them as possible. Hence, we view them as hard. LP$^{\text{MLN}}$ does not have a direct analog of rule (4) but, it seems natural to represent it by two rules:

$$ln(0.3)\ :\ loc(1) = r_1 \leftarrow broken, move \tag{7}$$

and

$$ln(0.7)\ :\ loc(1) = r_2 \leftarrow broken, move \tag{8}$$

where the logarithms are added to the probabilities to cancel the exponentiation from the definition of unnormalized measure w_M. In addition, the hard rule

$$\alpha :\ \leftarrow not\ loc(1) = r_1, not\ loc(1) = r_2 \tag{9}$$

is added to force $loc(1)$ to take a value (in P-log this is guaranteed by the semantics of disjunction). This concludes construction of M^0. It is worth noting that M_0 is similar to the program obtained from Π_0 by a general translation from P-log to LP$^{\text{MLN}}$ described in [Lee and Wang, 2016].

We will show that there is a simple relationship between probabilistic models defined by Π^0 and M^0. The possible worlds of Π^0 correspond to the possible worlds M^0 with non-zero probability (also called *probabilistic stable models* of M^0 in [Lee *et al.*, 2015]). Moreover, probability functions P_{M^0} and P_{Π^0} coincide on probabilistic stable models of M^0.

Let us first notice that $W_0 = \{move, loc(0) = r_1, loc(1) = r_2\}$ is an ASP model of $M^0_{W_0}$ and hence is a possible world of M^0. The probability of W_0 is 1. Clearly, W_0 is the only possible world of M^0 satisfying all its hard rules. M^0 however has other possible worlds. For instance, $V = \{move\}$ satisfies all the rules of $M^0 \setminus \{(1), (3), (9)\}$ and is the stable model of this program. Therefore, it is a possible world of M^0. It is easy to check however that V is not a probabilistic stable model of M^0. In fact, this is a consequence of a general result in [Lee *et al.*, 2015] which says that if there is a possible world of LP$^{\text{MLN}}$ program M which satisfies all its hard rules then every probabilistic stable model of M also satisfies them. The result clearly implies that W_0 is the only probabilistic stable model of M^0.

The program $M^1 = M^0 \cup \{\alpha\ :\ broken\}$ is again similar to Π^1. It has two probabilistic stable models, W_1 and W_2 with probabilities equal to 0.7 and 0.3 respectively. As before, there are other possible worlds but none of them satisfies all the hard rules of M^1 and hence they have probability 0.

A more serious difference can be observed however between the program Π^2 and the new program M^2 obtained from M^0 by adding the rule

$$\alpha : loc(1) = r_1 \tag{10}$$

Since the rules of Π^2 are strict and, therefore, should be satisfied by possible worlds, the program Π^2 is inconsistent. M^2 however does not have such a restriction. It will have three probabilistic stable models. The first one is an ASP model of $M^2 \setminus \{(2)\}$. It resolves contradiction by assuming that the robot failed to move. The second is an ASP model of $M^2 \setminus \{(3)\}$. The contradiction is removed by abandoning the causal law (3). Another possible explanation may given by an ASP model of $M^2 \setminus \{(10)\}$ which assumes that our observation of the robot being in r_1 after the execution of $move$ is incorrect. This seems to be a reasonable answer.

Finally, to model the effect of consistency restoring rule (5), we extend M^0 with the rule

$$w : broken \tag{11}$$

where w is a very large negative weight. The resulting program M^0_{new} will have 3 probabilistic stable models, in two of which the robot is believed to be broken, however the probabilities of both of them are very low. This behavior is quite different (and, in some sense, less elegant) from the similar case in P-log where the cr-rule (5) was not used since the program Π^0 is consistent. Similarly to Π^2_{new}, the program M^2_{new} consisting of M^0_{new} and the fact $\{loc(1) = r_1\}$ has exactly one probabilistic stable model W_2 (which satisfies all the hard rules of the program). As in P-log, the semantics of LP$^{\text{MLN}}$ allow updating of the probability of the robot to be in room r_2 at step 1 from 0 to 1 by adding new information. However, unlike in P-log, extending the original program M^0 with a soft counterpart (11) of the cr-rule (5) leads to introducing new probabilistic stable models of the program with negligible probabilities.

4 From LP$^{\text{MLN}}$ to P-log

In this section we state the main result of this paper: establishing a relationship between LP$^{\text{MLN}}$ and P-log programs.

First we need a definition. Let M be an LP$^{\text{MLN}}$ program and $At(M)$ be the set of atoms in M.

Definition 1 (Counterpart)**.** *A P-log program Π is called a counterpart of M if there exists a bijection ϕ from the set of probabilistic stable models of M to the set of possible worlds of Π such that*

1. *for every probabilistic stable model W of M , if P_M and P_Π are probability functions defined by M and Π respectively, then $P_M(W) = P_\Pi(\phi(W))$*
2. *for every probabilistic stable model W of M, $W \equiv_{At(M)} \phi(W)$, that is, W and $\phi(W)$ coincide on the atoms of M.*

Main Theorem. *For every LP^{MLN} program M (as defined in section 3) there exists its P-log counterpart, $\tau(M)$, which is linear-time constructible.* □

The previous theorem immediately implies the following important corollary:

Corollary 1. *If A is atom of an LP^{MLN} program M, then*

$$P_M(A) = P_{\tau(M)}(A)$$

where the probabilities $P_M(A)$ and $P_{\tau(M)}(A)$ are defined as the sum of probabilities of possible worlds which contain A of the corresponding program. □

To prove the theorem we need the following Lemma which gives an alternative characterization of probabilistic stable models of M.

Lemma 1. *Let W be an interpretation satisfying n hard rules of M. W is a probabilistic stable model of M if and only if*

1. *W is a possible world of M supported by some $M^0 \subseteq M$;*
2. *no possible world of M supported by some $M^1 \subseteq M$ satisfies more than n hard rules of M.*

□

Proof of the Main Theorem: Due to space limitations, we only provide a construction of $\tau(M)$ given a program M and define a map ϕ from definition 1 (the complete proof of the theorem, including a proof of lemma 1 and a short outline, can be found in the extended version of the paper[2]).

We will assume that all atoms in Π are of the form p, where p is an identifier (in the general case, the atoms of the form $p(t_1, \ldots, t_n)$ can be translated into unique identifiers).

In what follows we will construct a P-log program which chooses a subprogram M^0 of M and computes ASP models of M supported by M^0 such that no possible world of M is supported by a program containing more hard rules than M_0. By Lemma 1, they will be probabilistic stable models of M. Appropriate probability atoms will ensure that the corresponding probabilities match.

Let $r_1, \ldots, r_n$ be the enumeration of rules of M. We will refer to i as the *label* of r_i.

The translation $\tau(M)$ is defined as follows:

1. $\tau(M)$ contains
   (a) declarations of the sorts $hard$ and $soft$ – sets of labels of hard and soft rules of M respectively.
   (b) declaration $a : boolean$ for each atom a in the signature of Π.
   (c) declarations of the auxiliary attributes

$$h, b, selected, sat : soft \rightarrow boolean$$

$$ab : hard \rightarrow boolean$$

   whose meaning will be explained later.

   We refer to this part of the translation as the *declaration part* of τ.

[2] http://www.depts.ttu.edu/cs/research/krlab/pdfs/papers/mlplogfull.pdf

2. For every hard rule r_i of the form

$$\alpha : head \leftarrow body \tag{12}$$

$\tau(M)$ contains the rules:

$$head \leftarrow body, not\ ab(i) \tag{13}$$

$$ab(i) \overset{+}{\leftarrow} . \tag{14}$$

The auxiliary relation $ab(i)$ says that "rule r_i is abnormal (or not-applicable))". The addition of $not\ ab(i)$ turns the translation (13) of Π's rule (12) into a default rule of P-log. The cr-rule (14), called Contingency Axiom [Gelfond and Rushton, 2010], says that, the reasoner may possibly believe $ab(i)$. This possibility, however, may be used only if there is no way to obtain a consistent set of beliefs by using only regular rules of the program. It is commonly used to capture indirect exceptions to defaults [Gelfond and Kahl, 2014]. Together, these rules allow to stop the application of a minimal number of the hard rules of M thus avoiding possible inconsistency and conforming to the semantics of such rules in LP$^{\text{MLN}}$.

This completes the translation for programs consisting of hard rules only.

3. For every soft rule r_i of the form

$$w : head \leftarrow body \tag{15}$$

$\tau(M)$ contains the rules:

$$head \leftarrow body, selected(i) \tag{16}$$

$$random(selected(i)) \tag{17}$$

$$\leftarrow \neg selected(i), sat(i) \tag{18}$$

The auxiliary relation $selected(i)$ says that "the rule with label i is selected"; relation $sat(i)$ stands for 'the rule with label i is satisfied'. The addition of $selected(i)$ to the body of the translation (16) of M's rule (15) together with random selection rule (17) allows a reasoner to select soft rules of a candidate subprogram M_0 of M.

Constraint (18) is used to ensure that computed models of M_0 satisfy condition 1 from the Lemma 1. Of course, to make this work we need the definition of sat which is given by the following rules:

$$sat(i) \leftarrow b(i), h(i) \tag{19}$$

$$sat(i) \leftarrow not\ b(i) \tag{20}$$

$$b(i) \leftarrow B \tag{21}$$

where B is the body of soft rule r_i, and

$$h(i) \leftarrow l \tag{22}$$

for every literal l in the head of soft rule r_i. As expected, $b(i)$ stands for 'the body of r_i is satisfied' and $h(i)$ for 'the head of r_i is satisfied'.

4. Finally, for every $selected(i)$, $\tau(M)$ contains probability atom:

$$pr(selected(i)) = \frac{e^{w_i}}{1 + e^{w_i}} \tag{23}$$

which says 'the soft rule r_i with weight w_i is selected (that is, added to M^0) with probability $\frac{e^{w_i}}{1+e^{w_i}}$.

It is easy to see that the size of $\tau(M)$ is linear in terms of the size of M. Moreover, $\tau(M)$ is modular, that is, it can be easily extended if new rules are added to M.

The map ϕ is defined is follows:

$$\begin{aligned}\phi(W) = & W \cup \{ab(i) \mid i \in hard, W \text{ does not satisfy } r_i\} \\ & \cup \{sat(i) \mid i \in soft, W \text{ satisfies } r_i\} \\ & \cup \{selected(i) \mid i \in soft, W \text{ does not satisfy } r_i\} \\ & \cup \{\neg selected(i) \mid i \in soft, W \text{ satisfies } r_i\} \\ & \cup \{b(i) \mid i \in soft, W \text{ satisfies the body of } r_i\} \\ & \cup \{h(i) \mid i \in soft, W \text{ satisfies the head of } r_i\}\end{aligned}$$

The rest of the proof can be outlined as follows. We first need to show that for every probabilistic stable model W, $\phi(W)$ is a possible world of $\tau(M)$. This can be done by using standard techniques suitable for CR-Prolog programs. After that, we show the bijectivity of ϕ. This step can be split into two parts. Firstly, the surjectivity of $\phi(W)$ follows from the fact that a probabilistic stable model V of M obtained from a possible world W of $\tau(\Pi)$ by dropping all newly introduced literals from W satisfies $\phi(V) = W$. Secondly, the injectivity follows trivially from the definition of ϕ. Finally, the required probability equality from definition 1 follows from the definition of probabilistic functions in both languages. □

The following is an example of the translation.

Example. Consider the following LP$^{\text{MLN}}$ program M from [Lee *et al.*, 2015]:

$\alpha : concertBooked.$
$\alpha : longDrive \leftarrow concertBooked, not\ cancelled.$
$ln(0.2) : cancelled.$
$ln(0.8) :\leftarrow cancelled.$

The program has two probabilistic stable models (each of which satisfy both its hard rules):

1. $V_1 = \{concertBooked, cancelled\}$
2. $V_2 = \{concertBooked, longDrive\}$

with corresponding probabilities equal to 0.2 and 0.8.

The corresponding translation $\tau(M)$ looks as follows:

% declaration part:
$soft = \{3, 4\}.$
$hard = \{1, 2\}.$
$concertBooked, cancelled, longDrive : boolean.$
$b, h, selected, sat : soft \rightarrow boolean.$
$ab : hard \rightarrow boolean.$
% translation of hard rules:
$concertBooked \leftarrow not\ ab(1).$
$longDrive \leftarrow concertBooked, not\ cancelled, not\ ab(2).$

$ab(R) \stackrel{+}{\leftarrow}$.
% translation of soft rules:
$cancelled \leftarrow selected(3)$.
$\leftarrow cancelled, selected(4)$.
$random(selected(R))$.
$\leftarrow \neg selected(R), sat(R)$.
% definition of satisfiability:
$sat(R) \leftarrow not\ b(R)$.
$sat(R) \leftarrow b(R), h(R)$.
$b(3)$.
$b(4) \leftarrow cancelled$.
$h(3) \leftarrow cancelled$.
% probability atoms:
$pr(selected(3)) = 0.2/(1 + 0.2)$.
$pr(selected(4)) = 0.8/(1 + 0.8)$.

The translation $\tau(M)$ has two possible worlds:

1. $U_1 = \{selected(3), \neg selected(4), h(3), cancelled, b(3), b(4), sat(3), concertBooked\}$
2. $U_2 = \{\neg selected(3), selected(4), b(3), longDrive, concertBooked, sat(4)\}$

As expected, on the atoms of M, U_1 and U_2 coincide with the corresponding probabilistic stable models $\{cancelled, concertBooked\}$ and $\{concertBooked, longDrive\}$ of M (more specifically, $U_1 = \phi(V_1)$ and $U_2 = \phi(V_2)$). It can be easily checked that, as promised, $P_{\tau(M)}(U_1) = P_M(V_1) = 0.2$ and $P_{\tau(M)}(U_2) = P_M(V_2) = 0.8$.

5 Probabilities of Soft Rules in LP$^{\text{MLN}}$

Let M be an LP$^{\text{MLN}}$ program with at least one soft rule r_i of the form $w_i : head \leftarrow body$. The authors of LP$^{\text{MLN}}$ view r_i as an implication and define the probability $P_M(r_i)$ as follows:

$$P_M(r_i) = \sum_{W \in \Omega_M, W \models r_i} P_M(W) \tag{24}$$

Note that replacing Ω_M with the set of all probabilistic stable models of M gives an equivalent definition. We will use the result obtained in the previous section to investigate the relationship between the reasoner's confidence in r_i, i.e, its weight w_i and its probability $P_M(r_i)$.

It seems natural to assume that $P_M(r_i)$ would be proportional to w. However, this is not necessarily the case. Let us consider the following program:

$ln(3) :\ a$.
$ln(3) :\ \leftarrow a$.
$ln(2) :\ b$.

Despite the larger weight, the first rule has smaller probability than the third one (the corresponding probabilities are equal to $1/2$ and $2/3$ respectively).

Informally speaking, this happens because the first rule is inconsistent with the second one, while the third one doesn't have such a restriction.

We next use the results from the previous section to obtain an alternative understanding of the probability $P_M(r_i)$. Let $\tau(M)$ be the counterpart of M described there and ϕ be the bijection from Definition 1. It can be easily seen that r_i is satisfied by a possible world W of M iff $\phi(W)$ contains $selected(i)$. This, together with the first clause of Definition 1 implies that:

$$P_M(r_i) = P_{\tau(M)}(selected(i)) \tag{25}$$

That is, the probability of r_i in M is equal to the probability of $selected(i)$ in P-log program $\tau(M)$. In general, this probability depends on all possible worlds of $\tau(M)$ and their probabilities. However, for some cases it can be determined uniquely by the weight of r_i. This is always the case if $\tau(M)$ belongs to the class of coherent P-log programs, where this probability is equal to the value of the pr-atom $pr(selected(i))$ in (23). This class, and the sufficient conditions for a P-log program to be in it, are given in [Baral *et al.*, 2009].

For instance, it can be checked that the translation of the program M^3 consisting of soft facts $ln(3) : a$ and $ln(2) : b$ is coherent. Thus, the fact that probability of a is equal to $e^{ln(3)}/(1 + e^{ln(3)}) = 3/4$ can be obtained directly from the corresponding pr-atom (23) of $\tau(M^3)$. Note that, in general, to compute the probability of an atom, we may need to perform fairly complex inference (e.g, compute possible worlds of the program).

6 Conclusion and Future Work

We have defined a linear-time constructible modular translation τ from LP$^{\text{MLN}}$ programs into P-log programs. Non-zero probability possible worlds of an LP$^{\text{MLN}}$ program M coincide with possible worlds of $\tau(M)$ on atoms of M. Moreover, the probabilistic functions defined by M and $\tau(M)$ coincide on atoms M. The work allowed us to better understand both languages, including their treatment of potential inconsistencies, and opened a way to the development of LP$^{\text{MLN}}$ solvers based on P-log inference engines. We also believe that this work, together with the new complementary results from [Lee and Wang, 2016], will allow to use the theory developed for one language to discover properties of the other.

Our plans for future work are as follows.

1. We plan to complete the current work on the development of an efficient inference engine for P-log and use the translation τ to turn it into a solver for LP$^{\text{MLN}}$.
2. In the near future, we expect the appearance of LP$^{\text{MLN}}$ solver based on the algorithm from Section 3.4 [Lee *et al.*, 2015]. It will be interesting to use the translatiom from [Lee and Wang, 2016] to turn it into P-log solver and compare its performance with that of the one mentioned above.
3. We plan to investigate the possibility of adapting inference methods developed for MLN[Gogate and Domingos, 2012] and LP$^{\text{MLN}}$ for improving efficiency of P-log solvers.

Acknowledgments

We would like to thank Joohyung Lee and Yi Wang for useful discussions on the topic.

References

[Balduccini and Gelfond, 2003] Marcello Balduccini and Michael Gelfond. Logic programs with consistency-restoring rules. In *International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, volume 102. The AAAI Press, 2003.

[Baral *et al.*, 2004] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. In *Logic Programming and Nonmonotonic Reasoning*, pages 21–33. Springer, 2004.

[Baral *et al.*, 2009] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9(01):57–144, 2009.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[De Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467, 2007.

[Fierens *et al.*, 2015] 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*, 15(03):358–401, 2015.

[Gelfond and Kahl, 2014] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[Gelfond and Rushton, 2010] Michael Gelfond and Nelson Rushton. Causal and probabilistic reasoning in p-log. *Heuristics, Probabilities and Causality. A tribute to Judea Pearl*, pages 337–359, 2010.

[Gogate and Domingos, 2012] Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. *arXiv preprint arXiv:1202.3724*, 2012.

[Lee and Wang, 2016] Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2016.

[Lee *et al.*, 2015] Joohyung Lee, Yunsong Meng, and Yi Wang. Markov logc style weighted rules under the stable model semantics. In *Technical Communications of the 31st International Conference on Logic Programming*, 2015.

[Marek and Truszczynski, 1999] Victor W. Marek and Miroslaw Truszczynski. *Stable Models and an Alternative Logic Programming Paradigm*, pages 375–398. The Logic Programming Paradigm: a 25-Year Perspective. Springer Verlag, Berlin, 1999.

[Niemela, 1998] Ilkka Niemela. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. In *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 72–79, Jun 1998.

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

[Sato, 1995] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *In Proceedings of the 12th International Conference on Logic Programming (ICLP95*. Citeseer, 1995.

[Sato, 2009] Taisuke Sato. Generative modeling by prism. In *Logic Programming*, pages 24–35. Springer, 2009.

[Vennekens *et al.*, 2004] Joost Vennekens, Sofie Verbaeten, and Maurice Bruynooghe. Logic programs with annotated disjunctions. In *Logic Programming*, pages 431–445. Springer, 2004.

[Vennekens *et al.*, 2009] Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. Cp-logic: A language of causal probabilistic events and its relation to logic programming. *TPLP*, 9(3):245–308, 2009.

[Zhu, 2012] Weijun Zhu. *Plog: Its algorithms and applications*. PhD thesis, Texas Tech University, 2012.