Reasoning and Learning with Probabilistic Answer Set Programming

by

Yi Wang

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved March 2019 by the
Graduate Supervisory Committee:

Joohyung Lee, Chair
Chitta Baral
Subbarao Kambhampati
Sriraam Natarajan
Siddharth Srivastava

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Knowledge Representation (KR) is one of the prominent approaches to Artificial Intelligence (AI) that is concerned with representing knowledge in a form that computer systems can utilize to solve complex problems. Answer Set Programming (ASP), based on the stable model semantics, is a widely-used KR framework that facilitates elegant and efficient representations for many problem domains that require complex reasoning.

However, while ASP is effective on deterministic problem domains, it is not suitable for applications involving quantitative uncertainty, for example, those that require probabilistic reasoning. Furthermore, it is hard to utilize information that can be statistically induced from data with ASP problem modeling.

This dissertation presents the language $\mathrm{LP}^{\mathrm{MLN}}$, which is a probabilistic extension of the stable model semantics with the concept of weighted rules, inspired by Markov Logic. An $\mathrm{LP}^{\mathrm{MLN}}$ program defines a probability distribution over "soft" stable models, which may not satisfy all rules, but the more rules with the bigger weights they satisfy, the bigger their probabilities. $\mathrm{LP}^{\mathrm{MLN}}$ takes advantage of both ASP and Markov Logic in a single framework, allowing representation of problems that require both logical and probabilistic reasoning in an intuitive and elaboration tolerant way.

This dissertation establishes formal relations between $\mathrm{LP}^{\mathrm{MLN}}$ and several other formalisms, discusses inference and weight learning algorithms under $\mathrm{LP}^{\mathrm{MLN}}$, and presents systems implementing the algorithms. $\mathrm{LP}^{\mathrm{MLN}}$ systems can be used to compute other languages translatable into $\mathrm{LP}^{\mathrm{MLN}}$. The advantage of $\mathrm{LP}^{\mathrm{MLN}}$ for probabilistic reasoning is illustrated by a probabilistic extension of the action language $\mathcal{BC}$+, called $p\mathcal{BC}$+, defined as a high-level notation of $\mathrm{LP}^{\mathrm{MLN}}$ for describing transition systems. Various probabilistic reasoning about transition systems, especially probabilistic diagnosis, can be modeled in $p\mathcal{BC}$+ and computed using $\mathrm{LP}^{\mathrm{MLN}}$ systems.

$p\mathcal{BC}+$ is further extended with the notion of utility, through a decision-theoretic extension of $\mathrm{LP}^{\mathrm{MLN}}$, and related with Markov Decision Process (MDP) in terms of policy optimization problems. $p\mathcal{BC}+$ can be used to represent (PO)MDP in a succinct and elaboration tolerant way, which enables planning with (PO)MDP algorithms in action domains whose description requires rich KR constructs, such as recursive definitions and indirect effects of actions.

To my parents

ACKNOWLEDGMENTS

During my long journey to attain my Ph.D, I have received support from many people. First of all, I would like to thank my advisor, Dr. Joohyung Lee. I first met Dr. Lee at his Introduction to Theoretical Computer Science course. As a beginner to the field of "theoretical" computer science, and an auditing student at that point, I was impressed by the clarity of his lecture. Subsequently, I decided to take several more courses from Dr. Lee later and eventually became his Ph.D student. Dr. Lee's patient supervision has greatly improved my academic skills, especially helping me develop a strong pursuit of technical clarity and soundness. Six years spent as part of Dr. Lee's group has changed my personality as a researcher, from one who's reluctant to express their own opinions to one who can speak out ideas confidently. I am also grateful for being given countless opportunities to present at conferences as well as industrial interships. The style of Dr. Lee's supervision allowed me to be productive at research while still having time to enjoy hobbies. Thank you Dr. Lee, I enjoyed being your Ph.D student.

Next, I would like to thank the other students in Dr. Lee's group. Those who are more senior than me, Yunsong Meng, Michael Bartholomew, Joseph Babb and Yu Zhang, helped grow my passion for computer science and AI with inspiring discussions and words of encouragement. Those who joined the group later than me, Zhun Yang, Jiaxuan Pang and Man Luo, offered great help in improving my work with valuable comments, and kept my life as a Ph.D student pleasant and cheerful with various out-of-lab activities. I feel privileged to work with all these brilliant and kind people.

I would like to also thank my committee members, Chitta Baral, Subbarao Kambhampati, Sriraam Natarajan and Siddharth Srivastava, for their helpful comments and suggestions.

Finally, I would like to say thank you to my parents from the bottom of my heart for being supporting and encouraging no matter what decision I make for my life. My father kindled my passion for computer science at the very beginning by showing me and letting me play with a real computer when I was 4 years old. He also taught me the basics of programming and the binary number system when I was in primary school, in a way that is so lively and entertaining, which turned abstract mathematical concepts into something even a small child can enjoy. Interactions with my father throughout my life shaped my critical thinking ability as well as my passion for intellectual pursuits.

My work in this thesis was partially supported by the National Science Foundation under Grants IIS-1526301, IIS-1319794 and IIS-1815337.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Knowledge Representation (KR) is one of the prominent approaches to Artificial Intelligence (AI). It solves problems in AI by creating representations of the problem domain in a form that can facilitate automated reasoning about the problem domain. Once the representation is created, the solutions of the problem can be derived automatically from the semantics of the underlying formalism.

Answer Set Programming (ASP) is a widely-used KR framework that can facilitate compact and intuitive representations for many problem domains that require commonsense reasoning. The problem domain is encoded as an answer set program, so that the "answer sets" of the program, which can be found automatically by ASP solvers, correspond to the solutions of the problem. The nonmonotonicity of the underlying semantics of ASP, the stable model semantics, enables various types of reasoning including defeasible reasoning, causal reasoning, diagnostic reasoning, etc., many of which are hard to be modeled with SAT based logic formalisms. Useful knowledge representation constructs and efficient solvers allow ASP to handle various combinatorial search and commonsense reasoning problems in knowledge intensive domains elegantly and efficiently.

However, difficulty remains when it comes to domains with quantitative uncertainty, for example, reasoning tasks that involve probabilistic inference. The fact that ASP does not distinguish answer sets that are more likely to be true limits its application domains. Furthermore, due to the "crisp" nature of the stable model semantics, it is hard to utilize information that can be statistically induced from data with ASP problem modeling, since data is noisy in most real-world situations.

On the other hand, Markov Logic (Domingos and Lowd (2009)) is a prominent approach in Statistical Relational Learning (SRL), aimed at combining probabilistic graphical models and logic. The idea is to assign machine-learnable weights to logic formulas, so that a model of the logic theory does not have to satisfy all formulas, and the weight scheme induces a probability distribution over all models. However, since Markov Logic is based on standard first-order semantics, it is weak for commonsense reasoning. For example, causality and inductive definitions are hard to be succinctly represented with Markov Logic.

To overcome the difficulty in modeling quantitative uncertainty with ASP, we propose the language $\mathrm{LP}^{\mathrm{MLN}}$, which is a probabilistic extension of ASP that combines the advantages of ASP and Markov Logic in a single framework. In $\mathrm{LP}^{\mathrm{MLN}}$, we introduce the notion of weighted rules under the stable model semantics, following the log-linear models of Markov Logic. $\mathrm{LP}^{\mathrm{MLN}}$ allows representations of commonsense reasoning problems that require both logical and probabilistic reasoning in an intuitive and elaboration tolerant way. Furthermore, thanks to its close relation to Markov Logic, some learning methods developed for Markov Logic can be adapted for $\mathrm{LP}^{\mathrm{MLN}}$ learning, in this way bringing machine learning algorithms in the context of a KR formalism.

The relation between $\mathrm{LP}^{\mathrm{MLN}}$ and Markov Logic is analogous to the known relationship between ASP and SAT, in that the former follows the stable model semantics and the latter follows a standard SAT semantics. The relationship between $\mathrm{LP}^{\mathrm{MLN}}$ and ASP is analogous to the relationship between Markov Logic and SAT, in that the former is a weighted extension of the latter. Figure 1.1 summarizes the relationships between ASP, Markov Logic, SAT and $\mathrm{LP}^{\mathrm{MLN}}$.

In this dissertation, we define the syntax and semantics of $\mathrm{LP}^{\mathrm{MLN}}$, and discuss its formal relations to other formalisms such as ASP, Markov Logic, ProbLog, P-

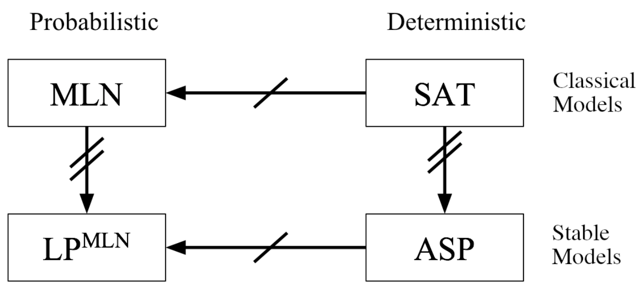


Figure 1.1: The Relation between ASP, Markov Logic, SAT and LP$^{\text{MLN}}$

log and Pearl's Causal Model (PCM). Based on the relationships, we implemented two systems LPMLN2ASP and LPMLN2MLN for LP$^{\text{MLN}}$ inference. The LP$^{\text{MLN}}$ inference systems can be used to compute other languages that are translatable into LP$^{\text{MLN}}$. We present the LP$^{\text{MLN}}$ weight learning system and illustrate how we can learn weights for probabilistic extensions of knowledge-rich domains that involve reachability analysis and reasoning about action dynamics, where ASP has been useful in the deterministic case.

To illustrate LP$^{\text{MLN}}$'s capability of reasoning in action domains, we present a probabilistic extension of the action language $\mathcal{BC}+$, called $p\mathcal{BC}+$, which is a high-level notation of LP$^{\text{MLN}}$ for describing transition systems. We show how probabilistic reasoning about dynamic domains, such as prediction and postdiction, as well as probabilistic diagnosis, can be modeled in the probabilistic language and computed using LP$^{\text{MLN}}$ inference and learning system.

In many probabilistic decision problems, the goal is to find a decision choice that yields the maximum expected utility. We extend LP$^{\text{MLN}}$ with the notion of utility, resulting in Decision-Theoretic LP$^{\text{MLN}}$ (DT-LP$^{\text{MLN}}$). based on DT-LP$^{\text{MLN}}$, we introduce the notion of utility in $p\mathcal{BC}+$, and further define the policy optimization problem in the context of $p\mathcal{BC}+$. We show that the policy optimization problem in terms of $p\mathcal{BC}+$ coincide with that of Markov Decision Process (MDP). We demonstrate that $p\mathcal{BC}+$ can be used to represent (PO)MDP in a succinct and elaboration tolerant way,

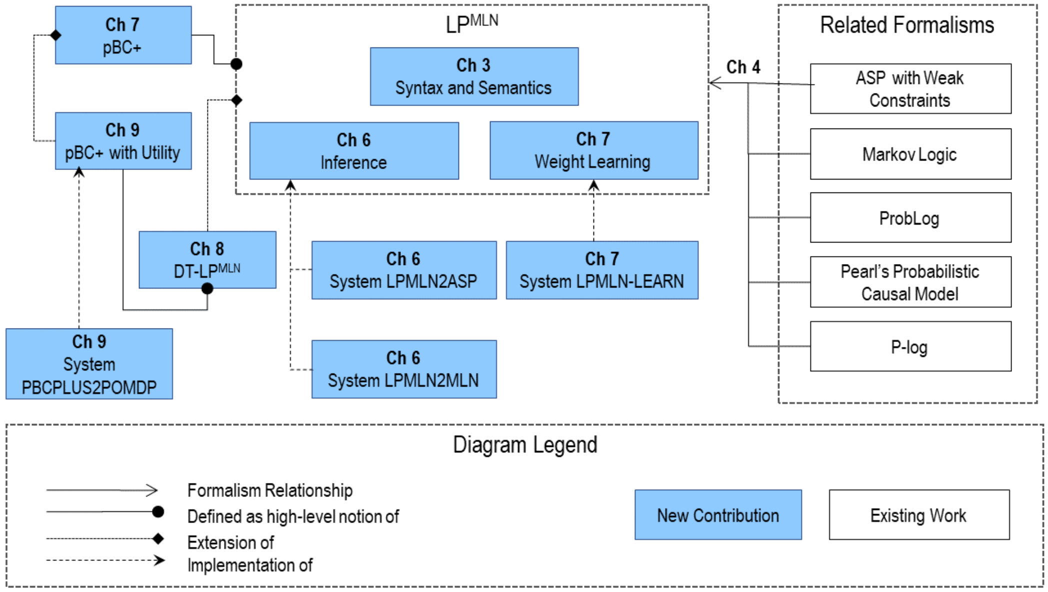


Figure 1.2: Dissertation Outline

which enables planning with (PO)MDP algorithms in action domains whose description requires rich KR constructs, such as recursive definitions and indirect effects of actions.

In these initial works on LP$^{\text{MLN}}$, our focus is on defining the language, studying its theoretical properties and exploring its expressivity. Computational efficiency is not the focus of this dissertation and the systems presented here are prototypical. Improving the efficiency of the systems and conducting empirical study of LP$^{\text{MLN}}$ in real-world applications are important future works. Figure 1.2 summarizes the contributions from this dissertation and indicates where each contribution is presented in this dissertation.

The rest of this dissertation is organized as follows: Chapter 2 reviews necessary background information, including the stable model semantics and Markov Logic. Chapter 3 defines the syntax and semantics of language LP$^{\text{MLN}}$. Chapter 4 discusses the formal relationship between LP$^{\text{MLN}}$ and related formalisms. In Chapter 5 and

Chapter 6 we discuss inference and learning methods for LP^{MLN}, respectively. Chapter 7 presents the action language $p\mathcal{BC}+$ defined in terms of LP^{MLN}. Chapter 8 presents the decision-theoretic extension of LP^{MLN}. Based on the decision-theoretic extension, Chapter 9 presents an extension of $p\mathcal{BC}+$ where policy optimization problem can be defined, and relates it with Markov Decision Process. We conclude in Chapter 10 with prospective contributions.

Chapter 2

BACKGROUND

Throughout this paper, we assume a first-order signature σ that has finitely many Herbrand interpretations.

2.1 Stable Model Semantics

A *rule* over σ is of the form

$$A_1;\dots;A_k \;\leftarrow\; A_{k+1},\dots,A_m, not\ A_{m+1},\dots, not\ A_n, not\ not\ A_{n+1},\dots, not\ not\ A_p \tag{2.1}$$

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

We will often identify (2.1) with the implication:

$$A_1 \vee \cdots \vee A_k \;\leftarrow\; A_{k+1} \wedge \ldots \wedge A_m \wedge \neg A_{m+1} \wedge \ldots \wedge \neg A_n \wedge \neg\neg A_{n+1} \wedge \ldots \wedge \neg\neg A_p\ . \tag{2.2}$$

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 (2.2) 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 (2.2) 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

$I \models_{\rm SM} \Pi$) 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

$$P \leftarrow Q \qquad Q \leftarrow P \qquad P \leftarrow not\ R \qquad R \leftarrow not\ P \tag{2.3}$$

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.

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 are proved to be useful in many KR domains.

2.2 Weak Constraints

Weak constraint (Buccafurri *et al.* (2000); Calimeri *et al.* (2012)) is a simple extension of answer set programs for expressing quantitative preference among answer sets. It is a part of ASP Core 2 language and has turned out to be useful in many practical applications. It is implemented in standard ASP solvers such as CLINGO and DLV. In Section 4.1.3, we will show that LP$^{\rm MLN}$ programs can be turned into ASP programs with weak constraints. In this section, we review the syntax and semantics of weak constraints.

A *weak constraint* has the form

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

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

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

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

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

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

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

The input language of CLINGO allows non-ground weak constraints that contain tuples of terms.

2.3 Markov Logic

The following is a review of Markov Logic 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

$$W_{\mathbb{L}}(I) = exp\left(\sum_{\substack{w:F \,\in\, \mathbb{L} \\ F \,\in\, \mathbb{L}_I}} w \right). \tag{2.4}$$

The probability of I under $\mathbb{L}$, denoted $P_{\mathbb{L}}(I)$, is defined as

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

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

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. For non-ground MLN, (2.4) can be written as

$$W_{\mathbb{L}}(I) = exp\left(\sum_{w_i:F_i \,\in\, \mathbb{L}} w_i m_i(I) \right)$$

where $m_i(I)$ is the number of ground instances of $w_i : F_i$ in $\mathbb{L}_I$.

2.4 Markov Decision Process

Markov Decision Process (MDP) provides a mathematical framework for modeling sequential decision making in domains where the effects of actions can be stochastic. In Chapter 9, we will relate the probabilistic action language $p\mathcal{BC}+$, which is defined in terms of LP^{MLN}, to MDP, showing that the finite horizon policy optimization problem under $p\mathcal{BC}+$ and MDP coincide, and $p\mathcal{BC}+$ can be used to represent MDP in a succinct and elaboration tolerant way.

A *Markov Decision Process (MDP)* M is a tuple $\langle S, A, T, R\rangle$ where (i) S is a set of states; (ii) A is a set of actions; (iii) $T : S \times A \times S \rightarrow [0, 1]$ defines transition probabilities; (iv) $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function.

2.4.1 Finite Horizon Policy Optimization

Given a nonnegative integer m as the maximum timestamp, and a history

$$\langle s_0, a_0, s_1, \dots, s_{m-1}, a_{m-1}, s_m\rangle$$

such that each $s_i \in S$ ($i \in \{0, \dots, m\}$) and each $a_i \in A$ ($i \in \{0, \dots, m-1\}$), the *total reward* of the history under MDP M is defined as

$$R_M(\langle s_0, a_0, s_1, \dots, s_{m-1}, a_{m-1}, s_m\rangle) = \sum_{i=0}^{m-1} R(s_i, a_i, s_{i+1}).$$

The probability of $\langle s_0, a_0, s_1, \dots, s_{m-1}, a_{m-1}, s_m\rangle$ under MDP M is defined as

$$P_M(\langle s_0, a_0, s_1, \dots, s_{m-1}, a_{m-1}, s_m\rangle) = \prod_{i=0}^{m-1} T(s_i, a_i, s_{i+1}).$$

A *non-stationary policy* $\pi : S \times ST \mapsto A$ is a function from $S \times ST$ to A, where $ST = \{0, \dots, m-1\}$. Given an initial state s_0, the *expected total reward* of a non-stationary policy π under MDP M is

$$\begin{aligned} ER_M(\pi, s_0) &= \underset{\substack{\langle s_1,\ldots,s_m\rangle: \\ s_i \in S \text{ for } i \in \{1,\ldots,m\}}}{E} [R_M(\langle s_0, \pi(s_0, 0), s_1, \ldots, s_{m-1}, \pi(s_{m-1}, m-1), s_m\rangle)] \\ &= \sum_{\substack{\langle s_1,\ldots,s_m\rangle: \\ s_i \in S \text{ for } i \in \{1,\ldots,m\}}} \Big(\sum_{i=0}^{m-1} R(s_i, \pi(s_i, i), s_{i+1})\Big) \times \Big(\prod_{i=0}^{m-1} T(s_i, \pi(s_i, i), s_{i+1})\Big). \end{aligned}$$

The *finite horizon policy optimization* problem is to find a non-stationary policy π that maximizes its expected total reward, given an initial state s_0, i.e., $\underset{\pi}{\text{argmax}}\, ER_M(\pi, s_0)$.

2.4.2 Infinite Horizon Policy Optimization

Policy optimization with the infinite horizon is defined similar to the finite horizon, except that a discount rate for the reward is introduced, and the policy is stationary, i.e., no need to mention time steps (ST). Given an infinite sequence of states and actions $\langle s_0, a_0, s_1, a_1, \ldots\rangle$, such that each $s_i \in S$ and each $a_i \in A$ ($i \in \{0, \ldots\}$), and a discount factor $\gamma \in [0, 1]$, the *discounted total reward* of the sequence under MDP M is defined as

$$R_M(\langle s_0, a_0, s_1, a_1, \ldots\rangle) = \sum_{i=0}^{\infty} \gamma^{i+1} R(s_i, a_i, s_{i+1}).$$

Various algorithms for MDP policy optimization have been developed, such as value iteration (Bellman (1957)) for an exact solution, and Q-learning (Watkins (1989)) for approximate solutions.

Chapter 3

LANGUAGE LP$^{\text{MLN}}$

3.1 Syntax

The syntax of LP$^{\text{MLN}}$ defines a set of weighted rules, which can be viewed as a special case of the syntax of MLNs by identifying rules with implications. More precisely, an LP$^{\text{MLN}}$ program Π 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 (2.1) and w is either a real number, or a symbol α for the "infinite weight", in which case we call the rule "hard rule".

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

We define $\overline{\Pi}$ to be the logic program obtained from Π by disregarding weights, i.e., $\overline{\Pi} = \{R \mid w : R \in \Pi\}$. We also use Π^{soft} and Π^{hard} to denote the subset of Π that consists of non-hard rules and hard rules, respectively.

3.2 Semantics

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

$$W_\Pi(I) = \begin{cases} exp\left(\sum\limits_{w:R \,\in\, \Pi_I} w \right) & \text{if } I \text{ is a stable model of } \overline{\Pi}_I; \\ 0 & \text{otherwise.} \end{cases} \quad (3.1)$$

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

$$P_\Pi(I) = \lim_{\alpha \to \infty} \frac{W_\Pi(I)}{\sum_{J \in PW} W_\Pi(J)},$$

where PW is the set of all Herbrand interpretations of σ. We say that I is a *(probabilistic) stable model* of Π if $P_\Pi(I) \neq 0$. We use SM[Π] to denote the set of interpretations I of Π such that I is a (deterministic) stable model of $\overline{\Pi}_I$. Similar to MLN, the definition is extended to any non-ground LP$^{\text{MLN}}$ program by identifying it with its *ground instance*. For non-ground LP$^{\text{MLN}}$, (3.1) can be written as

$$W_\Pi(I) = \begin{cases} exp\left(\sum\limits_{w_i:F_i \,\in\, \Pi} w_i m_i(I) \right) & \text{if } I \text{ is a stable model of } \overline{\Pi}_I; \\ 0 & \text{otherwise} \end{cases} \quad (3.2)$$

where $m_i(I)$ is the number of ground instances of $w_i : F_i$ in Π_I.

The intuition here is similar to that of Markov Logic. For each possible world I, we try to find a maximal subset (possibly empty) of $\overline{\Pi}$ for which I is a stable model (under the standard stable model semantics). In other words, the LP$^{\text{MLN}}$ semantics is similar to the MLN semantics except that the possible worlds are the *stable* models of some maximal subset of $\overline{\Pi}$, and the probability distribution is over these stable models. Unlike MLNs, such a subset may not necessarily exist, which means that no subset can account for the stability of the model. In that case, the weight of the interpretation is assigned 0. In the other case (when such a subset exists), it does not seem obvious if there is a *unique* maximal subset that accounts for the stability of I. Nevertheless, it follows from the following proposition that this is indeed the case, and that the unique maximal subset is exactly Π_I.

Proposition 1. *For any logic program Π and any subset Π' of Π, if I is a stable model of Π' and I satisfies Π, then I is a stable model of Π as well.*

The proposition tells us that if I is a stable model of a program, adding more rules to this program does not affect that I is a stable model of the resulting program as long as I satisfies the rules added. On the other hand, it is clear that I is no longer a stable model if I does not satisfy at least one of the rules added.

Consider an LP$^{\text{MLN}}$ program Π, whose deterministic part $\overline{\Pi}$ is the same as (2.3).

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

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

I	Π_I	$Pr_\Pi[I]$
$\emptyset$	$\{r_1, r_2\}$	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 (deterministic) stable models $\{P, Q\}$ and $\{R\}$ of $\overline{\Pi}$ are the (probabilistic) stable models of Π with the highest probability. In addition, Π has two other (probabilistic) stable models, which do not satisfy some rules in $\overline{\Pi}$ and is thus less probable.

It is easy to observe the following facts.

Proposition 2. *For any* $\mathrm{LP}^{\mathrm{MLN}}$ *program* Π, *(i) every (probabilistic) stable model of* Π *is an (MLN) model of* Π*; (ii) every stable model of* $\overline{\Pi}$ *is a (probabilistic) stable model of* Π.

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

Fierens *et al.* (2013) remark that "Markov Logic has the drawback that it cannot express (non-ground) inductive definitions." This is not the case for $\mathrm{LP}^{\mathrm{MLN}}$ as the following example illustrates. The example also shows how the Generate-Define-Test way of organizing rules in ASP can be applied to $\mathrm{LP}^{\mathrm{MLN}}$.

Example 1. *Consider a probabilistic variant of the Hamiltonian Cycle Problem, in which the presence of directed edges is probabilistic (which may be statistically induced from data). We are interested in the probabilities of potential Hamiltonian cycles, which are induced by the probabilities of the edges that participate in forming the cycle. This problem can be modeled by the following* $\mathrm{LP}^{\mathrm{MLN}}$ *representation:*

% input data

$w_1: \ Edge(1,2)$

$w_2: \ Edge(2,3)$

$\dots$

% generate

$\alpha: \ \{In(x,y)\}^{\mathrm{ch}} \leftarrow Edge(x,y)$

% define

$\alpha: \ R(x) \leftarrow In(1,x)$

$\alpha: \ R(y) \leftarrow R(x), In(x,y)$

% test

$\alpha: \leftarrow In(x,y_1), In(x,y_2) \qquad (y_1 \neq y_2)$

$\alpha: \leftarrow In(x_1,y), In(x_2,y) \qquad (x_1 \neq x_2)$

$\alpha: \leftarrow not\ R(x), Vertex(x)$

($In(x,y)$ *means that the edge* (x,y) *is in the Hamiltonian cycle;* $R(x)$ *means that* x *is reachable from the initial vertex* 1*). All the hard rules are those that are familiar from standard ASP as given in Lifschitz (2008), which illustrates that* $\mathrm{LP}^{\mathrm{MLN}}$ *is a natural extension of standard ASP.*

The weight scheme of $\mathrm{LP}^{\mathrm{MLN}}$ provides a simple but effective way to resolve certain inconsistencies in ASP programs.

Example 2. *For example, consider the simple ASP knowledge base KB_1:*

$$\begin{array}{rcl} Bird(x) & \leftarrow & ResidentBird(x) \\ Bird(x) & \leftarrow & MigratoryBird(x) \\ & \leftarrow & ResidentBird(x), MigratoryBird(x). \end{array}$$

One data source KB_2 (possibly acquired by some information extraction module) says that Jo is a ResidentBird:

$$ResidentBird(Jo)$$

while another data source KB_3 states that Jo is a MigratoryBird:

$$MigratoryBird(Jo).$$

The data about Jo is actually inconsistent w.r.t. KB_1, so under the (deterministic) stable model semantics, the combined knowledge base $KB = KB_1 \cup KB_2 \cup KB_3$ is not so meaningful. On the other hand, it is still intuitive to conclude that Jo is likely a Bird, and may be a ResidentBird or a MigratoryBird. Such reasoning is supported in $\mathrm{LP}^{\mathrm{MLN}}$*, as follows.*

Viewing rules from KB_1, KB_2 and KB_3 as $\mathrm{LP}^{\mathrm{MLN}}$ *rules yields*

$$\begin{array}{lll r} KB_1 & \alpha: & Bird(x) \leftarrow ResidentBird(x) & (r1) \\ & \alpha: & Bird(x) \leftarrow MigratoryBird(x) & (r2) \\ & \alpha: & \leftarrow ResidentBird(x), MigratoryBird(x) & (r3) \\ KB_2 & \alpha: & ResidentBird(Jo) & (r4) \\ KB_3 & \alpha: & MigratoryBird(Jo) & (r5) \end{array}$$

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

I	Π_I	$W_\Pi(I)$	$P_\Pi(I)$
$\emptyset$	$\{r_1, r_2, r_3\}$	$e^{3\alpha}$	0
$\{R(Jo)\}$	$\{r_2, r_3, r_4\}$	$e^{3\alpha}$	0
$\{M(Jo)\}$	$\{r_1, r_3, r_5\}$	$e^{3\alpha}$	0
$\{B(Jo)\}$	$\{r_1, r_2, r_3\}$	0	0
$\{R(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_4\}$	$e^{4\alpha}$	$\frac{1}{3}$
$\{M(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_5\}$	$e^{4\alpha}$	$\frac{1}{3}$
$\{R(Jo), M(Jo)\}$	$\{r_4, r_5\}$	$e^{2\alpha}$	0
$\{R(Jo), M(Jo), B(Jo)\}$	$\{r_1, r_2, r_4, r_5\}$	$e^{4\alpha}$	$\frac{1}{3}$

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

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

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

$$KB'_2 \quad 2: \quad ResidentBird(Jo) \qquad (r4')$$

$$KB'_3 \quad 1: \quad MigratoryBird(Jo) \quad (r5')$$

Then the table changes as follows.

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

$P(Bird(Jo)) = (e^2+e^1)/(e^2+e^1+e^0) = 0.67+0.24$, *so it becomes less certain, though it is still very likely that we can conclude that Jo is a Bird.*

Notice that the weight changes not only affect the probability, but also the stable models (having non-zero probabilities) themselves: Instead of $\{R(Jo), M(Jo), B(Jo)\}$, *the empty set is a stable model of the new program.*

Assigning a different certainty level to each rule affects the probability associated with each stable model, representing how certain we can derive the stable model from the knowledge base. This could be useful as more incoming data reinforces the certainty levels of the information.

Conditional probability under Π is defined as usual. For propositions A and B,

$$P_\Pi(A \mid B) = \frac{\sum\limits_{I \in SM[\Pi], I \models A \wedge B} P_\Pi(I)}{\sum\limits_{I \in SM[\Pi], I \models B} P_\Pi(I)}.$$

In (3.1), the weight assigned to each stable model can be regarded as a "reward": the more rules are true in deriving the stable model, the larger weight is assigned to it. It is possible to reformulate the definition of weight in a "penalty" based way. More precisely, the penalty based weight of an interpretation I is defined as the exponentiated negative sum of the weights of the rules that are not satisfied by I (when I is a stable model of $\overline{\Pi_I}$). Let

$$W_\Pi^{\rm pnt}(I) = \begin{cases} exp\left(-\sum\limits_{w:R\,\in\,\Pi \text{ and } I\not\models R} w\right) & \text{if } I \in \mathrm{SM}[\Pi]; \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

and

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

Let TW_Π be the sum of weights of all rules in Π, i.e.,

$$TW_\Pi = exp\left(\sum_{w:R\in\Pi} w\right).$$

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

Theorem 1. *For any* $\mathrm{LP^{MLN}}$ *program* Π *and any interpretation* I*,*

$$W_\Pi(I) = TW_\Pi \times W_\Pi^{\rm pnt}(I) \qquad \textit{and} \qquad P_\Pi(I) = P_\Pi^{\rm pnt}(I).$$

Similarly, for non-ground $\mathrm{LP^{MLN}}$, (3.3) can be written as

$$W_\Pi(I) = \begin{cases} exp\left(\sum\limits_{w_i:F_i\,\in\,\Pi} - w_i n_i(I)\right) & \text{if } I \text{ is a stable model of } \overline{\Pi}_I; \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where $n_i(I)$ is the number of ground instance of $w_i : F_i$ in $\Pi \setminus \Pi_I$.

Although the penalty-based reformulation appears to be more complicated, it has a few desirable features. One of them is that adding a trivial rule does not affect the weight of an interpretation, which is not the case with the original definition. More importantly, this reformulation leads to a better translation of $\mathrm{LP^{MLN}}$ programs into answer set programs as we discuss in Section 4.1.3.

Often we are interested in stable models that satisfy all hard rules (hard rules encode definite knowledge), in which case the probabilities of stable models can be computed from the weights of the soft rules only, as described below.

Let $\mathrm{SM}'[\Pi]$ be the set

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

and let

$$W'_{\Pi}(I) = \begin{cases} exp\left(\sum\limits_{w:R\,\in\,(\Pi^{\mathrm{soft}})_I} w\right) & \text{if } I \in \mathrm{SM}'[\Pi]; \\ 0 & \text{otherwise,} \end{cases} \tag{3.5}$$

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

Notice the absence of $\lim\limits_{\alpha\to\infty}$ in the definition of $P'_{\Pi}[I]$. Also, unlike $P_{\Pi}(I)$, $\mathrm{SM}'[\Pi]$ may be empty, in which case $P'_{\Pi}(I)$ is not defined. Otherwise, the following proposition tells us that the probability of an interpretation can be computed by considering the weights of the soft rules only.

Proposition 3. *If* $\mathrm{SM}'[\Pi]$ *is not empty, for every interpretation* I, $P'_{\Pi}(I)$ *coincides with* $P_{\Pi}(I)$.

It follows from this proposition that if $\mathrm{SM}'[\Pi]$ is not empty, then every stable model of Π (with non-zero probability) should satisfy all hard rules in Π.

3.3 Multi-Valued Probabilistic Programs

We introduce a simple fragment of $\mathrm{LP}^{\mathrm{MLN}}$, called multi-valued probabilistic programs. It allows us to represent probability more naturally. For simplicity of the presentation, we will assume a propositional signature. An extension to first-order signatures is straightforward.

We assume that the propositional signature σ is constructed from "constants" and their "values." A *constant* c is a symbol that is associated with a finite set $Dom(c)$, called the *domain*. The signature σ is constructed from a finite set of constants, consisting of atoms $c = v$ [1] for every constant c and every element v in $Dom(c)$. If the domain of c is $\{\mathbf{f}, \mathbf{t}\}$ then we say that c is *Boolean*, and abbreviate $c{=}\mathbf{t}$ as c and $c{=}\mathbf{f}$ as $\sim c$.

We assume that constants are divided into *probabilistic* constants and *regular* constants. A multi-valued probabilistic program $\mathbf{\Pi}$ is a tuple $\langle PF, \Pi\rangle$, where

- PF contains *probabilistic constant declarations* of the following form:

$$p_1 : c{=}v_1 \mid \cdots \mid p_n : c{=}v_n \tag{3.6}$$

  one for each probabilistic constant c, where $\{v_1, \ldots, v_n\} = Dom(c)$, $v_i \neq v_j$, $0 \leq p_1, \ldots, p_n \leq 1$ and $\sum_{i=1}^{n} p_i = 1$. We use $M_{\mathbf{\Pi}}(c = v_i)$ to denote p_i. In other words, PF describes the probability distribution over each "random variable" c.

- Π is a set of rules of the form (2.1) such that no A_i among $A_1, \ldots, A_k$ is a probabilistic constant.

The semantics of such a program $\mathbf{\Pi}$ is defined as a shorthand for LP$^{\text{MLN}}$ program $T(\mathbf{\Pi})$ of the same signature as follows.

- For each probabilistic constant declaration (3.6), $T(\mathbf{\Pi})$ contains, for each $i = 1, \ldots, n$, (i) $ln(p_i) : c{=}v_i$ if $0 < p_i < 1$; (ii) $\alpha :\ c{=}v_i$ if $p_i = 1$; (iii) $\alpha : \leftarrow c{=}v_i$ if $p_i = 0$.

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

- For each rule in Π of form (2.1), $T(\mathbf{\Pi})$ contains

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

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

$$\alpha: \quad \bot \leftarrow c{=}v_1 \wedge c = v_2 \tag{3.7}$$

for all $v_1, v_2 \in Dom(c)$ such that $v_1 \neq v_2$. For each probabilistic constant c, $T(\mathbf{\Pi})$ also contains the existence of value constraint

$$\alpha: \quad \bot \leftarrow \neg \bigvee_{v \in Dom(c)} c{=}v \ . \tag{3.8}$$

This means that a regular constant may be undefined (i.e., have no values associated with it), while a probabilistic constant is always associated with some value.

Example 3. *The multi-valued probabilistic program*

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

is understood as shorthand for the LP^{MLN} *program*

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

We say an interpretation of $\mathbf{\Pi}$ is *consistent* if it satisfies the hard rules (3.7) for every constant and (3.8) for every probabilistic constant. For any consistent interpretation I, we define the set $TC(I)$ ("Total Choice") to be

$$\{c = v \mid c \text{ is a probabilistic constant such that } c = v \in I\}$$

and define

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

For any interpretation I, we define

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

and

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

The following proposition tells us that the probability of an interpretation can be computed from the probabilities assigned to probabilistic atoms.

Proposition 4. *For any multi-valued probabilistic program $\mathbf{\Pi}$ such that each p_i in* (3.6) *is positive for every probabilistic constant c, if* $\mathrm{SM}''[\mathbf{\Pi}]$ *is not empty, then for any interpretation I, $P''_{\mathbf{\Pi}}(I)$ coincides with $P_{T(\mathbf{\Pi})}(I)$.*

3.4 Proofs

3.4.1 Proof of Proposition 1

We use $I \models_{SM} \Pi$ to denote "the interpretation I is a (deterministic) stable model of the program Π".

The proof of Proposition 1 uses the following theorem, which is a special case of Theorem 2 in Lee and Meng (2011). Given an ASP program Π of signature σ and a subset Y of σ, we use $LF_{\Pi}(Y)$ to denote the loop formula of Y for Π.

Theorem 2. *Let Π be a program of a finite first-order signature σ with no function constants of positive arity, and let I be an interpretation of σ that satisfies Π. The following conditions are equivalent to each other:*

(a) $I \models_{\mathrm{SM}} \Pi$;

(b) for every nonempty finite subset Y of atoms formed from constants in σ, I satisfies $LF_{\Pi}(Y)$;

(c) for every finite loop Y of Π, I satisfies $LF_{\Pi}(Y)$.

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.*

Proof. For any subset L of σ, since I is a stable model of Π', by Theorem 2, I satisfies $LF_{\Pi'}(L)$, that is, I satisfies $L^{\wedge} \rightarrow ES_{\Pi'}(L)$. It can be seen that the disjunctive terms in $ES_{\Pi'}(L)$ is a subset of the disjunctive terms in $ES_{\Pi}(L)$, and thus $ES_{\Pi'}(L)$ entails $ES_{\Pi}(L)$. So I satisfies $L^{\wedge} \rightarrow ES_{\Pi}(L)$, which is $LF_{\Pi}(L)$, and since in addition we have $I \models \Pi$, I is a stable model of Π. $\square$

3.4.2 Proof of Proposition 2

Proposition 2 *For any* $\mathrm{LP}^{\mathrm{MLN}}$ *program Π, (i) every (probabilistic) stable model of Π is an (MLN) model of Π; (ii) every stable model of $\overline{\Pi}$ is a (probabilistic) stable model of Π.*

Proof. (i) For any interpretation I, let $W_{\Pi}^{\mathrm{MLN}}(I)$ denote the weight of I under Π under MLN semantics. It can be easily see that when I is a stable model of $\overline{\Pi}_I$, we have

$$W_{\Pi}^{\rm MLN}(I) = W_{\Pi}(I).$$

so if I is a (probabilistic) stable model of Π, then I is an (MLN) model of Π.

(ii) If I is a stable model of $\overline{\Pi}$, then $\overline{\Pi}_I = \overline{\Pi}$ and clearly I is a stable model of $\overline{\Pi}_I$. So $W_{\Pi}(I) \neq 0$. Since I satisfies all hard rules in Π, $P_{\Pi}(I) \neq 0$. $\square$

3.4.3 Proof of Theorem 1

Theorem 1 *For any* LP$^{\rm MLN}$ *program* Π *and any interpretation* I*,*

$$W_{\Pi}(I) = TW_{\Pi} \times W_{\Pi}^{\rm pnt}(I) \qquad \textit{and} \qquad P_{\Pi}(I) = P_{\Pi}^{\rm pnt}(I).$$

Proof. We first show that $W_{\Pi}(I) = TW_{\Pi} \times W_{\Pi}^{\rm pnt}(I)$. This is obvious when $I \notin {\rm SM}[\Pi]$.

When $I \in {\rm SM}[\Pi]$, we have

$$\begin{aligned}
W_{\Pi}(I) &= exp\Bigg(\sum_{w\,:\,F \in \Pi \text{ and } I \models F} w\Bigg) \\
&= exp\Bigg(\sum_{w:F\in\Pi} w - \sum_{w\,:\,F \in \Pi \text{ and } I \not\models F} w) \\
&= exp\Bigg(\sum_{w:F\in\Pi} w\Bigg) \cdot exp\Bigg(-\sum_{w\,:\,F \in \Pi \text{ and } I \not\models F} w\Bigg) \\
&= TW_{\Pi} \cdot exp\Bigg(-\sum_{w\,:\,F \in \Pi \text{ and } I \not\models F} w\Bigg) \\
&= TW_{\Pi} \times W_{\Pi}^{\rm pnt}(I).
\end{aligned}$$

Consequently,

$$\begin{aligned}
P_\Pi(I) &= \frac{W_\Pi(I)}{\sum_J W_\Pi(J)} \\
&= \frac{TW_\Pi \cdot W_\Pi^{\text{pnt}}(I)}{\sum_J TW_\Pi \cdot W_\Pi^{\text{pnt}}(J)} \\
&= \frac{W_\Pi^{\text{pnt}}(I)}{\sum_J W_\Pi^{\text{pnt}}(J)} \cdot \frac{TW_\Pi}{TW_\Pi} \\
&= \frac{W_\Pi^{\text{pnt}}(I)}{\sum_J W_\Pi^{\text{pnt}}(J)} \\
&= P_\Pi^{\text{pnt}}(I).
\end{aligned}$$

□

3.4.4 Proof of Proposition 3

Proposition 3 *If* $\mathrm{SM}'[\Pi]$ *is not empty, for every interpretation* I*,* $P'_\Pi(I)$ *coincides with* $P_\Pi(I)$*.*

Proof. For any interpretation I, by definition, we have

$$\begin{aligned}
P_\Pi(I) &= \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_{J\in SM[\Pi]} W_\Pi(J)} \\
&= \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_{J\models_{SM}\overline{\Pi_J}} exp(\sum_{w:F\in\Pi_J} w)}.
\end{aligned}$$

By definition, if an interpretation I belongs to $\mathrm{SM}'[\Pi]$, then I satisfies $\overline{\Pi^{hard}}$ and I is a stable model of $\overline{\Pi_I}$.

- Suppose $I \in \mathrm{SM}'[\Pi]$, which implies that I satisfies $\overline{\Pi^{hard}}$ and is a stable model of $\overline{\Pi_I}$. Then we have

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\Pi_I} w)}{\sum_{J\models_{SM}\overline{\Pi_J}} exp(\sum_{w:F\in\Pi_J} w)}.$$

Splitting the denominator into two parts: those J's that satisfy $\overline{\Pi^{hard}}$ and those that do not, and extracting the weights of formulas in $\overline{\Pi^{hard}}$, we have

$$P_{\Pi}(I) = \lim_{\alpha\to\infty} \frac{exp(|\Pi^{hard}|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_I\setminus\Pi^{hard}} w)}{HSAT + HUNSAT}.$$

where

$$HSAT = exp(|\Pi^{hard}|\cdot\alpha)\cdot \sum_{J\models_{SM}\overline{\Pi_J}:J\models\overline{\Pi^{hard}}} exp(\sum_{w:F\in\Pi_J\setminus\Pi^{hard}} w)$$

is the sum of the weights from interpretation J's that satisfies $\overline{\Pi^{\mathrm{hard}}}$, and

$$HUNSAT = \sum_{J\models_{SM}\overline{\Pi_J}:J\not\models\overline{\Pi^{hard}}} exp(|\Pi^{hard}\cap\Pi_J|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_J\setminus\Pi^{hard}} w)$$

is the sum of the weights from interpretation J's that do not satisfy $\overline{\Pi^{\mathrm{hard}}}$.

We divide both the numerator and the denominator by $exp(|\Pi^{hard}|\cdot\alpha)$.

$$P_{\Pi}(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\Pi_I\setminus\Pi^{hard}} w)}{\frac{HSAT}{exp(|\Pi^{hard}|\cdot\alpha)} + \frac{HUNSAT}{exp(|\Pi^{hard}|\cdot\alpha)}}$$

where

$$\frac{HSAT}{exp(|\Pi^{hard}|\cdot\alpha)} = \sum_{J\models_{SM}\overline{\Pi_J}:J\models\overline{\Pi^{hard}}} exp(\sum_{w:F\in\Pi_J\setminus\Pi^{hard}} w)$$

and

$$\frac{HUNSAT}{exp(|\Pi^{hard}|\cdot\alpha)} = \sum_{J\models_{SM}\overline{\Pi_J}:J\not\models\overline{\Pi^{hard}}} \frac{exp(|\Pi^{hard}\cap\Pi_J|\cdot\alpha)}{exp(|\Pi^{hard}|\cdot\alpha)}\cdot exp(\sum_{w:F\in\Pi_J\setminus\Pi^{hard}} w)$$

For $J\not\models\overline{\Pi^{hard}}$, we note $|\Pi^{hard}\cap\Pi_J|\le|\Pi^{hard}|-1$, so

$$\begin{aligned} P_{\Pi}(I) &= \frac{exp(\sum_{w:F\in\Pi_I\setminus\Pi^{hard}} w)}{\sum_{J\models_{SM}\overline{\Pi_J}:J\models\overline{\Pi^{hard}}} exp(\sum_{w:F\in\Pi_J\setminus\Pi^{hard}} w)} \\ &= P'_{\Pi}(I). \end{aligned}$$

- Suppose $I \notin \text{SM}'[\Pi]$, which implies that I does not satisfy $\overline{\Pi^{hard}}$ or is not a stable model of $\overline{\Pi_I}$. Let K be any interpretation in $\text{SM}'[\Pi]$. By definition, K satisfies $\overline{\Pi^{hard}}$ and K is a stable model of $\overline{\Pi_K}$.
  - Suppose I is not a stable model of $\overline{\Pi_I}$. Then by definition, $W_\Pi(I) = W'_\Pi(I) = 0$, and thus $P_\Pi(I) = P'_\Pi(I) = 0$.
  - Suppose I is a stable model of $\overline{\Pi_I}$ but I does not satisfy $\overline{\Pi^{hard}}$.

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\Pi_I} w)}{\sum_{J\models_{SM}\overline{\Pi_J}} exp(\sum_{w:F\in\Pi_J} w)}.$$

Since K satisfies $\overline{\Pi^{hard}}$, we have $\overline{\Pi^{hard}} \subseteq \overline{\Pi_K}$. By assumption we have that K is a stable model of $\overline{\Pi_K}$. We split the denominator into K and the other interpretations, which gives

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\Pi_I} w)}{exp(\sum_{w:F\in\Pi_K} w) + \sum_{J\neq K:J\models_{SM}\overline{\Pi_J}} exp(\sum_{w:F\in\Pi_J} w)}.$$

Extracting weights from the formulas in Π^{hard}, we have

$$\begin{aligned} P_\Pi(I) &= \lim_{\alpha\to\infty} \frac{exp(|\Pi^{hard}\cap\Pi_I|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_I\setminus\Pi^{hard}} w)}{exp(|\Pi^{hard}|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_K\setminus\Pi^{hard}} w) + \sum_{J\neq K:J\models_{SM}\overline{\Pi_J}} exp(\sum_{w:F\in\Pi_J} w)} \\ &\leq \lim_{\alpha\to\infty} \frac{exp(|\Pi^{hard}\cap\Pi_I|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_I\setminus\Pi^{hard}} w)}{exp(|\Pi^{hard}|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_K\setminus\Pi^{hard}} w)}. \end{aligned}$$

Since I does not satisfy $\overline{\Pi^{hard}}$, we have $|\Pi^{hard}\cap\Pi_I| \leq |\Pi^{hard}| - 1$, and thus

$$P_\Pi(I) \leq \lim_{\alpha\to\infty} \frac{exp(|\Pi^{hard}\cap\Pi_I|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_I\setminus\Pi^{hard}} w)}{exp(|\Pi^{hard}|\cdot\alpha)\cdot exp(\sum_{w:F\in\Pi_K\setminus\Pi^{hard}} w)} = 0 = P'_\Pi(I).$$

□

3.4.5 Proof of Proposition 4

Given a multi-valued probabilistic LP^MLN program $\mathbf{\Pi} = \langle PF, \Pi\rangle$, we use $\sigma^{pf}(\mathbf{\Pi})$ to denote the set of all probabilistic constants in $\mathbf{\Pi}$. It can be seen that, if we have $M_{\mathbf{\Pi}}(c = v) > 0$ for all constants c and $v \in Dom(c)$, then given a consistent interpretation I, we have $T(\mathbf{\Pi})^{\text{hard}} = UEC \cup \Pi \cup SINGLE$, where

$$UEC = \{\bot \leftarrow c = v_1 \wedge c = v_2 \mid c \text{ is a constants of } \sigma \text{ and } v_1, v_2 \in Dom(c), v_1 \neq v_2\} \cup \left\{\bot \leftarrow \neg \bigvee_{v \in Dom(c)} c = v \mid c \in \sigma^{pf}(\mathbf{\Pi})\right\},$$

and

$$SINGLE = \{c = v \mid M_{\mathbf{\Pi}}(c = v) = 1\},$$

and $(T(\mathbf{\Pi})^{\text{soft}})_I = TC(I) \setminus SINGLE$.

Lemma 1. *For any multi-valued probabilistic program $\mathbf{\Pi} = \langle PF, \Pi \rangle$, for which $SM''[\mathbf{\Pi}]$ is not empty and $M_{\mathbf{\Pi}}(c = v) > 0$ for all constants c and $v \in Dom(c)$, and any interpretation I, I belongs to $\text{SM}'[T(\mathbf{\Pi})]$ if and only if I belongs to $SM''[\mathbf{\Pi}]$.*

Proof. It can be seen that

$$\begin{aligned}&\overline{T(\mathbf{\Pi})^{\text{hard}}} \cup \overline{(T(\mathbf{\Pi})^{\text{soft}})_I}\\ &= \Pi \cup UEC \cup SINGLE \cup (TC(I) \setminus SINGLE).\end{aligned}$$

($\Rightarrow$) Suppose I belongs to $\text{SM}'[T(\mathbf{\Pi})]$. By definition, I satisfies $\overline{T(\mathbf{\Pi})^{hard}}$, which contains UEC. Obviously since I satisfies UEC, I is consistent. For those $c = v \in SINGLE$, it must be the case that $Dom(c) = \{v\}$. In this case, we have $c = v \in I$ since I is consistent. So $SINGLE \subseteq TC(I)$ and thus $SINGLE \cup (TC(I) \setminus SINGLE) = TC(I)$. So we have

$$\begin{aligned}&\overline{T(\mathbf{\Pi})^{\text{hard}}} \cup \overline{(T(\mathbf{\Pi})^{\text{soft}})_I}\\ &= \Pi \cup UEC \cup TC(I).\end{aligned}$$

and since I is a stable model of $\overline{T(\mathbf{\Pi})^{\text{hard}}} \cup \overline{(T(\mathbf{\Pi})^{\text{soft}})_I}$, I is a stable model of $\Pi \cup UEC \cup TC(I)$. It follows that I is a stable model of $\Pi \cup TC(I)$ since UEC contains constraints only. Since in addition we have I is consistent, I belongs to $SM''[\mathbf{\Pi}]$.

($\Leftarrow$) Suppose I belongs to $SM''[\mathbf{\Pi}]$. By definition, I is consistent, and I is a stable model of $\Pi \cup TC(I)$. Clearly I satisfies UEC since I is consistent. Since UEC contains constraints only, I is a stable model $\Pi \cup TC(I) \cup UEC$. For those $c = v \in SINGLE$, it must be the case that $Dom(c) = \{v\}$. In this case, we have $c = v \in I$ since I is consistent. So $SINGLE \subseteq TC(I)$ and thus $SINGLE \cup (TC(I) \setminus SINGLE) = TC(I)$. So we have

$$
\begin{aligned}
&\Pi \cup UEC \cup TC(I) \\
&= \Pi \cup UEC \cup SINGLE \cup (TC(I) \setminus SINGLE) \\
&= \overline{T(\mathbf{\Pi})^{\text{hard}}} \cup \overline{(T(\mathbf{\Pi})^{\text{soft}})_I}
\end{aligned}
$$

So I is a stable model of $\overline{T(\mathbf{\Pi})^{\text{hard}}} \cup \overline{(T(\mathbf{\Pi})^{\text{soft}})_I}$, and by definition I belongs to $\text{SM}'[T(\mathbf{\Pi})]$. $\square$

The following proposition establishes a useful property.

Proposition 5. *Given an* LP$^{\text{MLN}}$ *program* Π *such that* $SM'[\Pi]$ *is not empty, and an interpretation* I*, the following three statements are equivalent:*

1. *I is a stable model of Π;*
2. *$I \in SM'[\Pi]$;*
3. *$P'_{\Pi}(I) > 0$.*

Lemma 2. *For any multi-valued probabilistic program $\mathbf{\Pi} = \langle PF, \Pi \rangle$, for which $SM''[\mathbf{\Pi}]$ is not empty and $M_{\mathbf{\Pi}}(c = v) > 0$ for all constants c and $v \in Dom(c)$, and any interpretation I, I is a stable model of $T(\mathbf{\Pi})$ if and only if $I \in SM''[\mathbf{\Pi}]$.*

Proof. By Lemma 1, I belongs to $\mathrm{SM}'[T(\mathbf{\Pi})]$ if and only if I belong to $SM''[\mathbf{\Pi}]$. By Proposition 5, I is a stable model of $T(\mathbf{\Pi})$ if and only if $I \in \mathrm{SM}'[T(\mathbf{\Pi})]$. So I is a stable model of $T(\mathbf{\Pi})$ if and only if $I \in SM''[\mathbf{\Pi}]$. □

Lemma 2 does not hold when $M_{\mathbf{\Pi}}(c = v) = 0$ for some constant c and $v \in Dom(c)$.

Example 4. *Consider the following multi-valued probabilistic* LP$^{\mathrm{MLN}}$ $\mathbf{\Pi}$*:*

$$1 : c = 1 \mid 0 : c = 2$$

$$p$$

which translates into

$$\begin{array}{lll} \alpha & : & c = 1 \\ \alpha & : & \bot \leftarrow c = 2 \\ \alpha & : & p. \end{array}$$

The interpretation $I = \{c = 2, p\}$ belongs to the set $SM''[\mathbf{\Pi}]$. However, it is not a stable model of $T(\mathbf{\Pi})$, since one hard rule is violated.

Proposition 4 *For any multi-valued probabilistic program $\mathbf{\Pi}$ such that each p_i in* (3.6) *is positive for every probabilistic constant c, if $\mathrm{SM}''[\mathbf{\Pi}]$ is not empty, then for any interpretation I, $P''_{\mathbf{\Pi}}(I)$ coincides with $P_{T(\mathbf{\Pi})}(I)$.*

Proof.

- Suppose $I \in SM''[\mathbf{\Pi}]$. By Lemma 1, we have $I \in \mathrm{SM}'[\mathbf{\Pi}]$. By Proposition 3, we have

$$
\begin{aligned}
P_{T(\mathbf{\Pi})}(I) &= P'_{T(\mathbf{\Pi})}(I) \\
&= \frac{W'_{T(\mathbf{\Pi})}(I)}{\sum_{J \in SM'[T(\mathbf{\Pi})]} W'_{T(\mathbf{\Pi})}(J)} \\
&= \frac{exp(\sum_{w:R \in T(\mathbf{\Pi})_I} w)}{\sum_{J \in SM'[T(\mathbf{\Pi})]} exp(\sum_{w:R \in T(\mathbf{\Pi})_J} w)} \\
&= \frac{\prod_{w:R \in T(\mathbf{\Pi})_I} exp(w)}{\sum_{J \in SM'[T(\mathbf{\Pi})]} \prod_{w:R \in T(\mathbf{\Pi})_J} exp(w)} \\
&= \frac{\prod_{c \,\in\, \sigma^{pf}(\mathbf{\Pi}) \text{ and } c^I \,=\, v} M_{\mathbf{\Pi}}(c = v)}{\sum_{J \in SM'[T(\mathbf{\Pi})]} \prod_{c \,\in\, \sigma^{pf}(\mathbf{\Pi}) \text{ and } c^J \,=\, v} M_{\mathbf{\Pi}}(c = v)} \\
&= P''_{\mathbf{\Pi}}(I)
\end{aligned}
$$

- Suppose $I \notin SM''[\mathbf{\Pi}]$. By Lemma 2, I is not a stable model of $T(\mathbf{\Pi})$, so $P_{T(\mathbf{\Pi})}(I) = 0$. On the other hand, $P''_{\mathbf{\Pi}}(I) = 0$ since $W''_{\mathbf{\Pi}}(I) = 0$.

□

Chapter 4

RELATION TO OTHER FORMALISMS

LP$^{\text{MLN}}$ is a middle-ground language that connects to many other formalisms in KR and SRL. In this section, we discuss the formal relation between LP$^{\text{MLN}}$ and ASP, Markov Logic, ProbLog, P-log and Pearl's Causal Model (PCM). We show that these languages can be translated into LP$^{\text{MLN}}$, which means that all these seemingly very different formalisms are indeed related, and, practically, we can use an LP$^{\text{MLN}}$ implementations to compute these languages.

4.1 Relation to ASP

4.1.1 Turning ASP into LP$^{\text{MLN}}$

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

Theorem 3. *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.*

Note that when the ASP program Π is inconsistent, it does not have any (deterministic) stable model. However, the LP$^{\text{MLN}}$ program $\mathbb{P}_\Pi$ can still have (probabilistic) stable models, as example 2 indicates.

4.1.2 Weak Constraints and LP$^{\rm MLN}$

The idea of softening rules in LP$^{\rm MLN}$ is similar to the idea of weak constraints in ASP, which is used for certain optimization problems. In this section, we show that ASP programs with weak constraints can be translated into LP$^{\rm MLN}$ programs.

Since levels can be compiled into weights (Buccafurri *et al.* (2000)), we consider weak constraints of the form

$$:\sim \ Body \quad [Weight] \tag{4.1}$$

where *Weight* is a positive integer. We assume all weak constraints are grounded. The penalty of a stable model is defined as the sum of the weights of all weak constraints whose bodies are satisfied by the stable model.

Such a program can be turned into an LP$^{\rm MLN}$ program as follows. Each weak constraint (4.1) is turned into

$$-w : \quad \bot \leftarrow \neg Body.$$

The standard ASP rules are identified with hard rules in LP$^{\rm MLN}$. For example, the program with weak constraints

$$\begin{array}{ll} a \vee b & :\sim a \ \ [1] \\ c \leftarrow b & :\sim b \ \ [1] \\ & :\sim c \ \ [1] \end{array}$$

is turned into

$$\begin{array}{llll} \alpha : & a \vee b & -1 : & \bot \leftarrow \neg a \\ \alpha : & c \leftarrow b & -1 : & \bot \leftarrow \neg b \\ & & -1 : & \bot \leftarrow \neg c. \end{array}$$

The LP$^{\rm MLN}$ program has two stable models: $\{a\}$ with the normalized weight $\frac{e^{-1}}{e^{-1}+e^{-2}}$ and $\{b, c\}$ with the normalized weight $\frac{e^{-2}}{e^{-1}+e^{-2}}$. The former, with the larger normalized weight, is the stable model of the original program containing the weak constraints.

Proposition 6. *For any program with weak constraints that has a stable model, its stable models are the same as the stable models of the corresponding* LP$^{\text{MLN}}$ *program with the highest normalized weight.*

4.1.3 Turning LP$^{\text{MLN}}$ *into ASP with Weak Constraints*

In the paper by Balai and Gelfond (2016), it is shown that LP$^{\text{MLN}}$ programs can be turned into P-log. In this section, we show that using a similar translation, it is even possible to turn LP$^{\text{MLN}}$ programs into answer set programs with weak constraints.

We turn each (possibly non-ground) rule

$$w_i: \quad Head_i(\mathbf{x}) \leftarrow Body_i(\mathbf{x})$$

in an LP$^{\text{MLN}}$ program Π, where i is the index of the rule and $\mathbf{x}$ is the list of global variables in the rule, into ASP rules

$$\begin{array}{rcl} \texttt{sat}(i, w_i, \mathbf{x}) & \leftarrow & Head_i(\mathbf{x}) \\ \texttt{sat}(i, w_i, \mathbf{x}) & \leftarrow & \texttt{not}\ Body_i(\mathbf{x}) \\ Head_i(\mathbf{x}) & \leftarrow & Body_i(\mathbf{x}),\ \texttt{not not sat}(i, w_i, \mathbf{x}) \\ & :\sim & \texttt{sat}(i, w_i, \mathbf{x}). \quad [-w_i'@l, i, \mathbf{x}] \end{array} \tag{4.2}$$

where (i) $w_i' = 1$ and $l = 1$ if w_i is α; and (ii) $w_i' = w_i$ and $l = 0$ otherwise.[1]

Intuitively, a ground `sat` atom is true if the corresponding ground rule obtained from the original program is true. For each true `sat` atom, a weak constraint imposes on the stable model the opposite of the weight as a penalty, which can be viewed as imposing the weight as a reward.

By $\mathsf{lpmln2asp}^{\mathsf{rwd}}(\Pi)$ we denote the resulting ASP program containing weak constraints. The following theorem states the correctness of the translation.

[1] CLINGO restricts the weights in weak constraints to be integers only. To implement the translation using CLINGO, we need to turn w_i' into an integer by multiplying some factor.

Theorem 4. *For any* LP$^{\text{MLN}}$ *program* Π*, there is a 1-1 correspondence* ϕ *between* SM[Π] *and the set of stable models of* $\mathsf{lpmln2asp^{rwd}}(\Pi)$*, where*

$$\phi(I) = I \cup \{\mathtt{sat}(i, w_i, \mathbf{c}) \mid w_i : Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \textit{ in } gr_\sigma[\Pi], I \models Body_i(\mathbf{c}) \rightarrow Head_i(\mathbf{c})\}.$$

Furthermore,

$$W_\Pi(I) = exp\left(\sum_{\mathtt{sat}(i,w_i,\mathbf{c}) \in \phi(I)} w_i\right). \tag{4.3}$$

Also, ϕ *is a 1-1 correspondence between the most probable stable models of* Π *and the optimal stable models of* $\mathsf{lpmln2asp^{rwd}}(\Pi)$.

While the translation is simple and modular, there are a few problems with using this translation to compute LP$^{\text{MLN}}$ using ASP solvers. First, the translation does not necessarily yield a program that is acceptable in CLINGO and requires a further translation. In particular, the first and the second rules of (4.2) may not be in the syntax of CLINGO. (The third rule contains double negations, which are allowed in CLINGO from version 4.) Second, more importantly, when we translate non-ground LP$^{\text{MLN}}$ rules into the input language of ASP solvers, the first and the second rules of (4.2) may be unsafe, so CLINGO cannot ground the program. We now introduce an alternative translation that avoids these problems by basing on the penalty-based concept of weights.

Based on the reformulation of LP$^{\text{MLN}}$ weight (3.3), we introduce another translation that turns LP$^{\text{MLN}}$ programs into ASP programs. The translation ensures that a safe LP$^{\text{MLN}}$ program is always turned into a safe ASP program, and the resulting program is readily acceptable as an input to CLINGO.[2]

We define the translation $\mathsf{lpmln2asp^{pnt}}(\Pi)$ by translating each (possibly non-ground)

[2]An LP$^{\text{MLN}}$ program Π is *safe* if its unweighted program $\overline{\Pi}$ is safe as defined by Calimeri *et al.* (2012).

rule

$$w_i : \qquad Head_i(\mathbf{x}) \leftarrow Body_i(\mathbf{x})$$

in an LP$^{\text{MLN}}$ program Π, where i is the index of the rule and $\mathbf{x}$ is the list of global variables in the rule, into ASP rules

$$\begin{array}{rcl} \texttt{unsat}(i, w_i, \mathbf{x}) & \leftarrow & Body_i(\mathbf{x}),\ \texttt{not}\ Head_i(\mathbf{x}) \\ Head_i(\mathbf{x}) & \leftarrow & Body_i(\mathbf{x}),\ \texttt{not}\ \texttt{unsat}(i, w_i, \mathbf{x}) \\ & :\sim & \texttt{unsat}(i, w_i, \mathbf{x}). \quad [w_i'@l, i, \mathbf{x}] \end{array} \tag{4.4}$$

where (i) $w_i' = 1$ and $l = 1$ if w_i is α; and (ii) $w_i' = w_i$ and $l = 0$ otherwise.[3]

Intuitively, the first rule of (4.4) makes atom $\texttt{unsat}(i, w_i, \mathbf{x})$ true when the i-th rule in the original program is not satisfied. In that case, the second rule is not effective, and w_i is imposed on the penalty of the stable model. On the other hand, if the i-th rule is satisfied, atom $\texttt{unsat}(i, w_i, \mathbf{x})$ is false, the rule $Head_i(\mathbf{x}) \leftarrow Body_i(\mathbf{x})$ is effective, and the penalty is not imposed.

The following theorem is an extension of Corollary 2 by Lee and Yang (2017) to allow non-ground programs and to consider the correspondence between all stable models, not only the most probable ones.

Theorem 5. *For any* LP$^{\text{MLN}}$ *program* Π*, there is a 1-1 correspondence* ϕ *between* SM$[\Pi]$ *and the set of stable models of* $\mathsf{lpmln2asp}^{\mathsf{pnt}}(\Pi)$*, where*

$$\phi(I) = I \cup \{\texttt{unsat}(i, w_i, \mathbf{c}) \mid w_i : Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \textit{ in } gr_\sigma[\Pi], I \not\models Body_i(\mathbf{c}) \rightarrow Head_i(\mathbf{c})\}.$$

Furthermore,

$$W_\Pi^{\text{pnt}}(I) = exp\Bigg(- \sum_{\texttt{unsat}(i, w_i, \mathbf{c}) \in \phi(I)} w_i \Bigg). \tag{4.5}$$

Also, ϕ *is a 1-1 correspondence between the most probable stable models of* Π *and the optimal stable models of* $\mathsf{lpmln2asp}^{\mathsf{pnt}}(\Pi)$*.*

[3] In the case $Head_i$ is a disjunction $l_1; \dots, l_n$, expression $\texttt{not}\ Head_i$ stands for $\texttt{not}\ l_1, \dots, \texttt{not}\ l_n$.

Theorem 5, in conjunction with Theorem 1, provides a way to compute the probability of a stable model of an LP$^{\text{MLN}}$ program by examining the `unsat` atoms satisfied by the corresponding stable model of the translated ASP program.

4.2 Relation to Markov Logic

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

MLNs can be easily embedded in LP$^{\text{MLN}}$. More precisely, any MLN $\mathbb{L}$ whose formulas have the form (2.2) can be turned into an LP$^{\text{MLN}}$ program $\Pi_{\mathbb{L}}$ so that the models of $\mathbb{L}$ coincide with the stable models of $\Pi_{\mathbb{L}}$, keeping the same probability distribution.

LP$^{\text{MLN}}$ program $\Pi_{\mathbb{L}}$ is obtained from $\mathbb{L}$ by adding

$$w : \quad \{A\}^{\text{ch}}$$

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

Theorem 6. *For any MLN $\mathbb{L}$ whose formulas have the form* (2.2)*, $\mathbb{L}$ and $\Pi_{\mathbb{L}}$ have the same probability distribution over all interpretations, and consequently, the models of $\mathbb{L}$ and the stable models of $\Pi_{\mathbb{L}}$ coincide.*

The rule form restriction imposed in Theorem 6 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 $P \vee Q \leftarrow R$.

4.2.2 Turning LP$^{\text{MLN}}$ *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. This allows us to compute the stable models using SAT solvers. The method can be extended to $\mathrm{LP}^{\mathrm{MLN}}$ so that their stable models along with the probability distribution can be computed using existing implementations of MLNs, such as Alchemy [4] and Tuffy. [5]

We refer the reader to Ferraris *et al.* (2006) for the definitions of a loop L and a loop formula $LF_{\Pi}(L)$ for program Π consisting of rules of the form (2.1)

The following theorem tells us how the stable model semantics can be reduced to the standard propositional logic semantics, via the concept of loop formulas.

Theorem 7. *(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 (2.3) has loops $\{P\}, \{Q\}, \{R\}, \{P, Q\}$, and the corresponding disjunctive loop formulas are

$$\begin{array}{rcl} P & \rightarrow & Q \vee \neg R \\ R & \rightarrow & \neg P \\ Q & \rightarrow & P \\ P \wedge Q & \rightarrow & \neg R. \end{array} \tag{4.6}$$

The stable models $\{P, Q\}$, $\{R\}$ of (2.3) are exactly the models of (2.3) that satisfy (4.6).

We extend Theorem 7 to turn $\mathrm{LP}^{\mathrm{MLN}}$ programs Π into MLN programs. We define $\mathbb{L}_{\Pi}$ to be the union of Π and $\{\alpha : LF_{\overline{\Pi}}(L) \mid L \text{ is a loop of } \overline{\Pi}\}$.

Theorem 8. *For any* $\mathrm{LP}^{\mathrm{MLN}}$ *program* Π *such that*

$$\{R \mid \alpha : R \in \Pi\} \cup \{LF_{\overline{\Pi}}(L) \mid L \textit{ is a loop of } \overline{\Pi}\}$$

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

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

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

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

In the special case when the program is *tight* (that is, its dependency graph is acyclic), the size of loop formulas is linear in the size of input programs (Lee (2005)). In this case, loop formulas coincide with *completion*.

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

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

for each ground atom A.

This is a straightforward extension of the completion from Lee and Lifschitz (2003) by simply assigning the infinite weight α to the completion formulas.

Theorem 9. *For any tight* LP$^{\text{MLN}}$ *program* Π *such that* $\text{SM}'[\Pi]$ *is not empty,* Π *(under the* LP$^{\text{MLN}}$ *semantics) and* $Comp(\Pi)$ *(under the MLN semantics) have the same probability distribution over all interpretations.*

Theorem 9 is a special case of Theorem 8.

4.3 Relation to ProbLog

ProbLog is a well-developed probabilistic logic programming language that is based on the distribution semantics by Sato (1995). It is closely related to LP$^{\text{MLN}}$.

In this section, we show that LP$^{\text{MLN}}$ is a proper generalization of ProbLog.

4.3.1 Review: ProbLog

We review the version of ProbLog from Fierens *et al.* (2013). As before, we identify a non-ground ProbLog program with its ground instance. So for simplicity we restrict attention to ground ProbLog programs.

In ProbLog, ground atoms over σ are divided into two groups: *probabilistic* atoms and *derived* atoms. A *(ground) ProbLog program* $\mathbb{P}$ is a tuple $\langle PF, \Pi \rangle$, where

- PF is a set of ground probabilistic facts of the form $pr :: a$, where pr is a real number in $[0, 1]$, and a is a probabilistic atom, and
- Π is a set of ground rules of the form (2.1) such that $k = 1$ and $p = n$, and the head does not contain a probabilistic atom.

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

$$pr(a_1) \times \cdots \times pr(a_m) \times (1-pr(b_1)) \times \cdots \times (1-pr(b_n)),$$

where $b_1, \ldots, b_n$ are the probabilistic atoms not belonging to C, and each of $pr(a_i)$ and $pr(b_j)$ is the probability assigned to a_i and b_j according to the set PF of ground probabilistic atoms.

The ProbLog semantics is only well-defined for programs $\mathbb{P} = \langle PF, \Pi \rangle$ such that $C \cup \Pi$ has a "total" (two-valued) well-founded model for each possible total choice C. Given such $\mathbb{P}$, for each interpretation I, $P_{\mathbb{P}}(I)$ is defined as $P_{\mathbb{P}}(C)$ if there exists a total choice C such that I is the total well-founded model of $C \cup \Pi$, and 0 otherwise

4.3.2 *Embedding ProbLog in* LP$^{\text{MLN}}$

Given a ProbLog program $\mathbb{P} = \langle PF, \Pi \rangle$, we construct the corresponding LP$^{\text{MLN}}$ program $\Pi_{\mathbb{P}}$ as follows:

- For each probabilistic fact $pr :: a$ in $\mathbb{P}$, LP$^{\text{MLN}}$ program $\Pi_{\mathbb{P}}$ contains (i) $ln(pr) : a$ and $ln(1 - pr) : \leftarrow a$ if $0 < pr < 1$; (ii) $\alpha :\ a$ if $pr = 1$; (iii) $\alpha :\ \leftarrow a$ if $pr = 0$;
- For each rule $R \in \Pi$, $\Pi_{\mathbb{P}}$ contains $\alpha : R$. In other words, R is identified with a hard rule in $\Pi_{\mathbb{P}}$.

Theorem 10. *Any (well-defined) ProbLog program $\mathbb{P}$ and its* LP$^{\text{MLN}}$ *representation $\Pi_{\mathbb{P}}$ have the same probability distribution over all interpretations.*

Syntactically, LP$^{\text{MLN}}$ allows more general rules than ProbLog, such as disjunctions in the head, as well as the empty head and double negations in the body. Further, LP$^{\text{MLN}}$ allows rules to be weighted as well as facts, and 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$^{\text{MLN}}$ handles multiple stable models in a flexible way similar to the way MLNs handle multiple models.

On the other hand, Theorem 10 justifies using an implementation of ProbLog [6] to compute a fragment of LP$^{\text{MLN}}$.

4.4 Relation to Pearl's Causal Model

Both answer set programs and Probabilistic Causal Models (PCM) allow for representing causality, Baral and Hunsaker (2007) has shown how PCM can be embedded in P-log, a probabilistic extension of answer set programs. In this section, we follow

[6] http://dtai.cs.kuleuven.be/problog

a similar approach and show that PCM can be embedded in LP^{MLN}. This result can be viewed as a generalization of the result from Bochman and Lifschitz (2015), and it illustrates that LP^{MLN} is a natural probabilistic extension of answer set programs. While Baral and Hunsaker (2007) did not report any experiments, we show in Section 5.4.2 that PCM can be executed through an implementation of LP^{MLN}.

4.4.1 Review: Pearls' Probabilistic Causal Model

In this section, we review Pearl's Probabilistic Causal Model.

Notation: We use capital letters (e.g., X, Y, Z, U, V) for (lists of) atoms and lower case letters (x, y, z, u, v) for generic symbols for specific (lists of) truth values taken by the corresponding (lists of) atoms. We often write x to denote $X = x$.

As usual, a propositional formula is constructed from atoms, **t**, **f**, and propositional connectives, $\neg$, $\wedge$, $\vee$, $\rightarrow$.

Definition 1 (structural theory)**.** *Assume that the set of propositional atoms is partitioned into a set of of exogenous atoms U and a set of endogenous atoms $V = \{V_1, \dots, V_n\}$. A* Boolean structural theory *is $\langle U, V, F\rangle$, where F is a set of equations $V_i = F_i$, one for each endogenous atom V_i, and F_i is a propositional formula.*

Definition 2 (causal diagram)**.** *The* causal diagram *of a Boolean structural theory $\langle U, V, F\rangle$ is the directed graph whose vertices are the atoms in $U \cup V$ and an edge goes from V_j to V_i if there is an equation $V_i = F_i$ in the structural theory such that V_j occurs in F_i. We say that the structural theory is acyclic if its causal diagram is acyclic.*

For any interpretation I and J of $U \cup V$, we say that $J \neq^V I$ if J and I agree on all atoms in U and do not agree on some atoms in V.

Definition 3 (solution)**.** *Given a Boolean causal theory $\langle U, V, F\rangle$, a* solution *(or a* causal world*) I is any interpretation of $U \cup V$ such that*

- *I satisfies the equivalences $V_i \leftrightarrow F_i$ for all equations $V_i = F_i$ in F, and*
- *no other interpretation J such that $J \neq^V I$ satisfies all such equivalences $V_i \leftrightarrow F_i$.*

Definition 4 (causal model)**.** *A* (Boolean) causal model *$\langle U, V, F\rangle$ is an acyclic Boolean structural theory that has a unique solution for each realization (i.e., truth assignment) of U; in other words, each truth assignment of U has a unique expansion to $U \cup V$ that is a solution.*

Definition 5 (probabilistic causal model)**.** *A* Probabilistic (Boolean) Structural Theory *is a pair*

$$\langle\langle U, V, F\rangle, P(U)\rangle \tag{4.7}$$

where $\langle U, V, F\rangle$ is a Boolean structural theory, and $P(U)$ is a probability distribution over U. We assume that exogenous atoms are independent of each other. A Probabilisitic (Boolean) Causal Model (PCM) *is a probabilistic structural theory* (4.7) *such that $\langle U, V, F\rangle$ is a causal model. The* solution*s of PCM* (4.7) *are the solutions of $\langle U, V, F\rangle$. The probability of a solution I under the PCM $\mathbb{M}$, denoted $P_{\mathbb{M}}(I)$, is defined as $P(U = I_U)$, where I_U is a restriction of I over U.*

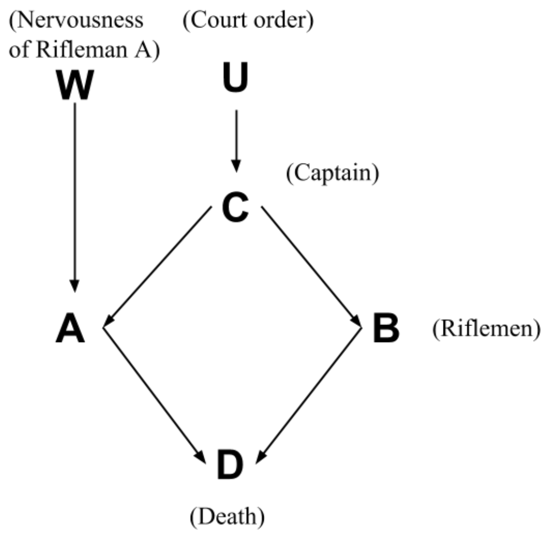
Given a PCM $\mathbb{M} = \langle\langle U, V, F\rangle, P(U)\rangle$, for any subset Y of V, we will write $Y_{\mathbb{M}}(u)$ to denote the truth assignment of Y in the solution of $\mathbb{M}$ induced by u. The probability of $Y = y$ is defined as

$$P_{\mathbb{M}}(Y = y) = \textstyle\sum_{\{u \mid Y_{\mathbb{M}}(u) = y\}} P(u).$$

For any subset Y, Z of V, $P_{\mathbb{M}}(Y = y \mid Z = z)$ is defined as

$$P_{\mathbb{M}}(Y = y \mid Z = z) = \frac{\sum_{\{u \ \mid \ Y_{\mathbb{M}}(u) \,=\, y \text{ and } Z_{\mathbb{M}}(u) \,=\, z\}} P(u)}{\sum_{\{u \mid Z_{\mathbb{M}}(u) = z\}} P(u)}.$$

Example 5. *Consider, for example, the causal model* $\mathbb{M}_{FS}$ *for the Firing Squad example from Section 7.1.2 of Pearl (2000):*



$\mathbb{M}_{FS}$:

$$C = U$$

$$A = C \vee W$$

$$B = C$$

$$D = A \vee B$$

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 causal model has four solutions for each realization of* U *and* W. *(*$W_{\mathbf{f}}$ *is shorthand for* $W = \mathbf{f}$. *Others are similar.)*

Solutions	*Probability*
$\{W_{\mathbf{f}}, U_{\mathbf{f}}, C_{\mathbf{f}}, A_{\mathbf{f}}, B_{\mathbf{f}}, D_{\mathbf{f}}\}$	$(1-p)(1-q)$
$\{W_{\mathbf{t}}, U_{\mathbf{f}}, C_{\mathbf{f}}, A_{\mathbf{t}}, B_{\mathbf{f}}, D_{\mathbf{t}}\}$	$(1-p)q$
$\{U_{\mathbf{t}}, W_{\mathbf{f}}, C_{\mathbf{t}}, A_{\mathbf{t}}, B_{\mathbf{t}}, D_{\mathbf{t}}\}$	$p(1-q)$
$\{U_{\mathbf{t}}, W_{\mathbf{t}}, C_{\mathbf{t}}, A_{\mathbf{t}}, B_{\mathbf{t}}, D_{\mathbf{t}}\}$	pq

Various queries involving probabilistic inference can be answered following the semantics. The following are some examples:

1. **Prediction** *Given that the court did not order the execution, what is probability that the prisoner is dead?*

According to the definition, we have

$$
\begin{aligned}
&P_{\mathbb{M}_{FS}}(D = True \mid U = False)\\
=&\frac{\sum_{\{u,w \mid D(u,w)=True\ and\ U(u,w)=False\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}{\sum_{\{u,w \mid U(u,w)=False\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}\\
=&\frac{P_{\mathbb{M}_{FS}}(U = False, W = True)}{P_{\mathbb{M}_{FS}}(U = False, W = True) + P_{\mathbb{M}_{FS}}(U = False, W = False)}\\
=&\frac{(1-p)q}{(1-p)q + (1-p)(1-q)}\\
=&q.
\end{aligned}
$$

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

   According to the definition, we have

$$
\begin{aligned}
&P_{\mathbb{M}_{FS}}(U = True \mid D = True)\\
=&\frac{\sum_{\{u,w \mid U(u,w)=True\ and\ D(u,w)=True\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}{\sum_{\{u,w \mid D(u,w)=True\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}\\
=&\frac{P_{\mathbb{M}_{FS}}(U = True, W = True) + P_{\mathbb{M}_{FS}}(U = True, W = False)}{P_{\mathbb{M}_{FS}}(U = True, W = True) + P_{\mathbb{M}_{FS}}(U = True, W = False) + P_{\mathbb{M}_{FS}}(U = False, W = True)}\\
=&\frac{pq + p(1-q)}{pq + p(1-q) + (1-p)q}\\
=&\frac{p}{1-(1-p)(1-q)}
\end{aligned}
$$

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

According to the definition, we have

$$
\begin{aligned}
&P_{\mathbb{M}_{FS}}(B = True \mid A = True)\\
=&\frac{\sum_{\{u,w \mid B(u,w)=True\ and\ A(u,w)=True\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}{\sum_{\{u,w \mid A(u,w)=True\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}\\
=&\frac{P_{\mathbb{M}_{FS}}(U = True, W = True) + P_{\mathbb{M}_{FS}}(U = True, W = False)}{P_{\mathbb{M}_{FS}}(U = True, W = True) + P_{\mathbb{M}_{FS}}(U = False, W = True) + P(U = True, W = False)}\\
=&\frac{pq + p(1-q)}{pq + (1-p)q + p(1-q)}\\
=&\frac{p}{1-(1-p)(1-q)}
\end{aligned}
$$

4. **Action** *Given that the captain did not signal the execution, what is the probability that the prisoner is dead if rifleman A decided to shoot?*

   According to the definition, we have

$$
\begin{aligned}
&P_{\mathbb{M}_{FS}}(D_{A=True} = True \mid C = False)\\
=&\frac{\sum_{\{u,w \mid D_{A=True}(u,w)=True\ and\ C(u,w)=False\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}{\sum_{\{u,w \mid C(u,w)=False\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}\\
=&\frac{P_{\mathbb{M}_{FS}}(U = False, W = True) + P_{\mathbb{M}_{FS}}(U = False, W = False)}{P_{\mathbb{M}_{FS}}(U = False, W = True) + P_{\mathbb{M}_{FS}}(U = False, W = False)}\\
=&1
\end{aligned}
$$

5. **Counterfactual** *Given that the prisoner is dead, what is the probability that the prisoner were not dead when A did not shoot?*

   According to the definition, we have

$$
\begin{aligned}
&P_{\mathbb{M}_{FS}}(D_{A=False} = False \mid D = True)\\
=&\frac{\sum_{\{u,w \mid D_{A=False}(u,w)=False\ and\ D(u,w)=True\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}{\sum_{\{u,w \mid D(u,w)=True\}} P_{\mathbb{M}_{FS}}(U = u, W = w)}\\
=&\frac{P_{\mathbb{M}_{FS}}(U = False, W = True)}{P_{\mathbb{M}_{FS}}(U = True, W = True) + P_{\mathbb{M}_{FS}}(U = True, W = False) + P_{\mathbb{M}_{FS}}(U = False, W = True)}\\
=&\frac{(1-p)q}{pq + p(1-q) + (1-p)q}\\
=&\frac{(1-p)q}{1-(1-p)(1-q)}.
\end{aligned}
$$

4.4.2 Embedding Pearl's Probabilistic Causal Model in LP$^{\text{MLN}}$

Since causal models assume propositional formulas, it is convenient to discuss the result by first extending the syntax of LP$^{\text{MLN}}$ to weighted propositional formulas, that is of the form $w : F$ where F is a propositional formula and w is either a real number or a symbol α. We refer the reader to Ferraris (2005) for the definition of a stable model for propositional formulas. Extending LP$^{\text{MLN}}$ to this general syntax is straightforward, which we skip due to lack of space.

Given a probabilistic causal model $\mathbb{M} = \langle\langle U, V, F\rangle, P\rangle$, we construct the corresponding LP$^{\text{MLN}}$ $\Pi_{\mathbb{M}}$ as follows. For simplicity, we assume that every variable in $U \cup V$ has Boolean domain, and all the functions in F involve only the logical operators $\wedge$ and $\neg$. Π_M contains the following atoms in the signature:

- an atom U_i for each exogenous variable $U_i \in U$;
- atoms of the form $V_i(w)$ where $w \in \{actual, counterfactual\}$, for each endogenous variable $V_i \in V$;
- atoms of the form $Do(val_{V_i}, w)$ where $w \in \{actual, counterfactual\}$ and $val \in \{True, False\}$, for each endogenous variable $V_i \in V$.

$\Pi_{\mathbb{M}}$ contains the following rules:

- rules

  $$ln(P(U_i = True)) : \quad U_i$$

  and

  $$ln(P(U_i = False)) : \quad \leftarrow U_i$$

  for all exogenous variable $U_i \in U$;

- a rule

$$\alpha : \quad V_i(w) \leftarrow f_i(w), \neg Do(True_{V_i}, w), \neg Do(False_{V_i}, w)$$

for every function F_i in F, of the form $V_i = f_i$, where $V_i \in V$ and f_i is a Boolean function on $U \cup V \setminus \{V_i\}$. By $f_i(w)$ we denote the Boolean formula obtained from f_i by replacing every endogenous variable V_i by $V_i(w)$;

- a rule

$$\alpha : \quad V_i(w) \leftarrow Do(True_{V_i}, w)$$

for every $V_i \in V$.

Theorem 11. *Given any $Y \subseteq V$ and variable assignments $X = x$, $Y = y$, $Z = z$, the probability defined by PCM, $P_{\mathbb{M}}(Y_{X=x} = y \mid Z = z)$, is equal to the following probability defined by* LP$^{\text{MLN}}$ *semantics,*

$$P_{\mathbb{M}}(Y_{X=x} = y \mid Z = z) = P_{\Pi_{\mathbb{M}}}(Do(X = x, counterfactual) \wedge Y(counterfacutual) = y \mid Z(actual) = z)$$
$$= \frac{\sum_{I \vDash Do(X=x, counterfactual) \wedge Y(counterfacutual)=y \wedge Z(actual)=z} P(I)}{\sum_{I \vDash Z(actual)=z} P(I)}$$

where $Do(X = x, counterfactual)$ is an abbreviation of

$$Do(x_{1X_1}, counterfactual) \wedge \cdots \wedge Do(x_{nX_n}, counterfactual)$$

for $X = \langle X_1, \ldots, X_n \rangle$ and $x = \langle x_1, \ldots, x_n \rangle$, and similarly $V(w) = v$ where V is Y or Z, v is y or z and w is $actual$ or $counterfactual$ is an abbreviation of

$$V_1(w) = v_1 \wedge \cdots \wedge V_n(w) = v_n$$

for $V = \langle V_1, \ldots, V_n \rangle$ and $v = \langle v_1, \ldots, v_n \rangle$.

Example 6. *The PCM $\mathbb{M}_{FS}$ in Example 5 corresponds to the following* LP$^{\text{MLN}}$ *pro-*

gram $\Pi_{(\mathbb{M}_{FS})}$ *($w \in \{actual, counterfactual\}$)*

$$
\begin{aligned}
ln(p) &: U \\
ln(1-p) &: \leftarrow U \\
ln(q) &: W \\
ln(1-q) &: \leftarrow W \\
\alpha &: C(w) \leftarrow U \wedge \neg Do(True_C, w) \wedge Do(False_C, w) \\
\alpha &: A(w) \leftarrow C(w) \wedge \neg Do(True_A, w) \wedge Do(False_A, w) \\
\alpha &: B(w) \leftarrow C(w) \wedge \neg Do(True_B, w) \wedge Do(False_B, w) \\
\alpha &: D(w) \leftarrow A(w) \wedge \neg Do(True_D, w) \wedge Do(False_D, w) \\
\alpha &: D(w) \leftarrow B(w) \wedge \neg Do(True_D, w) \wedge Do(False_D, w) \\
\alpha &: A(w) \leftarrow W \wedge \neg Do(True_A, w) \wedge Do(False_A, w) \\
\alpha &: C(w) \leftarrow Do(True_C, w) \\
\alpha &: A(w) \leftarrow Do(True_A, w) \\
\alpha &: B(w) \leftarrow Do(True_B, w) \\
\alpha &: D(w) \leftarrow Do(True_D, w)
\end{aligned}
$$

One can check that for the example queries listed in Example 5, under $\Pi_{(\mathbb{M}_{FS})}$ *we have*

- **Prediction**

$$
\begin{aligned}
&P_{\mathbb{M}_{FS}}(D = True \mid U = False) \\
=&P_{\Pi_{(\mathbb{M}_{FS})}}(D(counterfactual) = True \mid U = False) \\
=&q.
\end{aligned}
$$

- **Abduction**

$$\begin{aligned}&P_{\mathbb{M}_{FS}}(U = True \mid D = True)\\ =&P_{\Pi_{(\mathbb{M}_{FS})}}(U = True \mid D(actual) = True)\\ =&\frac{p}{1-(1-p)(1-q)}.\end{aligned}$$

- **Transduction**

$$\begin{aligned}&P_{\mathbb{M}_{FS}}(B = True \mid A = True)\\ =&P_{\Pi_{(\mathbb{M}_{FS})}}(B(counterfactual) = True \mid A(actual) = True)\\ =&\frac{p}{1-(1-p)(1-q)}.\end{aligned}$$

- **Action**

$$\begin{aligned}&P_{\mathbb{M}_{FS}}(D_{A=True} = True \mid C = False)\\ =&P_{\Pi_{(\mathbb{M}_{FS})}}(Do(True_A, counterfactual) \wedge D(counterfactual) = True \mid C(actual) = False)\\ =&1.\end{aligned}$$

- **Counterfactal**

$$\begin{aligned}&P_{\mathbb{M}_{FS}}(D_{A=False} = False \mid D = True)\\ =&P_{\Pi_{(\mathbb{M}_{FS})}}(Do(False_A, counterfactual) \wedge D(counterfacutual) = False \mid D(actual) = True)\\ =&\frac{(1-p)q}{1-(1-p)(1-q)}.\end{aligned}$$

4.5 Relation to P-log

Similar to $\mathrm{LP}^{\mathrm{MLN}}$, P-log (Baral *et al.* (2004)) is another probabilistic programming language whose logical foundation is the stable model semantics. Like $\mathrm{LP}^{\mathrm{MLN}}$, it adopts the stable model semantics as the logic component. However, P-log uses

Causal Bayesian Networks as the underlying probabilistic graphical model. P-log is distinct from other earlier work in that it allows for expressing probabilistic nonmonotonicity, the ability of the reasoner to change its probabilistic model as a result of new information. However, inference in the implementation of P-log is not scalable as it has to enumerate all stable models.

The following reviews the results from Lee and Wang (2016), which embeds a fragment of P-log in LP$^{\text{MLN}}$. For an embedding of complete P-log in LP$^{\text{MLN}}$, please refer to Lee and Yang (2017). The other direction, i.e., turning LP$^{\text{MLN}}$ into P-log turns out to be also possible, as shown by Balai and Gelfond (2016).

4.5.1 Simple P-log

In this section, we define a fragment of P-log, which we call *simple P-log*.

Syntax

Let σ be a multi-valued propositional signature as defined in Section 3.3. A simple P-log program Π is a tuple

$$\Pi = \langle R, S, P, Obs, Act \rangle \tag{4.8}$$

where

- R is a set of normal rules of the form

$$A \leftarrow B_1, \dots, B_m, not\ B_{m+1}, \dots, not\ B_n. \tag{4.9}$$

  Here and after we assume $A, B_1, \dots, B_n$ are atoms from σ ($0 \leq m \leq n$).

- S is a set of *random selection rules* of the form

$$[r]\ \ random(c) \leftarrow B_1, \dots, B_m, not\ B_{m+1}, \dots, not\ B_n \tag{4.10}$$

  where r is an identifier and c is a constant.

- P is a set of *probability atoms (pr-atoms)* of the form

$$pr_r(c{=}v \mid B_1, \dots, B_m, not\ B_{m+1}, \dots, not\ B_n) = p$$

  where r is the identifier of some random selection rule in S, c is a constant, and $v \in Dom(c)$, and $p \in [0,1]$.

- Obs is a set of atomic facts of the form $Obs(c = v)$ where c is a constant and $v \in Dom(c)$.

- Act is a set of atomic facts of the form $Do(c = v)$ where c is a constant and $v \in Dom(c)$.

Example 7. *We use the following simple P-log program as our main example ($d \in \{D_1, D_2\}$, $y \in \{1, \dots 6\}$):*

$$Owner(D_1){=}Mike$$
$$Owner(D_2){=}John$$
$$Even(d) \leftarrow Roll(d){=}y,\ \ y\ mod\ 2 = 0$$
$$\sim\!Even(d) \leftarrow not\ \ Even(d)$$
$$[r(d)]\, random(Roll(d))$$
$$pr(Roll(d){=}6 \mid Owner(d){=}Mike) = \tfrac{1}{4}.$$

Semantics

Given a simple P-log program Π of the form (4.8), a (standard) ASP program $\tau(\Pi)$ with the multi-valued signature σ' is constructed as follows:

- σ' contains all atoms in σ, and atom $Intervene(c){=}\mathbf{t}$ (abbreviated as $Intervene(c)$) for every constant c of σ; the domain of $Intervene(c)$ is $\{\mathbf{t}\}$.

- $\tau(\Pi)$ contains all rules in R.

- For each random selection rule of the form (4.10) with $Dom(c) = \{v_1, \ldots, v_n\}$, $\tau(\Pi)$ contains the following rules:

$$c\!=\!v_1; \ldots; c\!=\!v_n \leftarrow$$
$$B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n, not\ Intervene(c).$$

- $\tau(\Pi)$ contains all atomic facts in Obs and Act.

- For every atom $c\!=\!v$ in σ,

$$\leftarrow Obs(c\!=\!v), not\ c\!=\!v.$$

- For every atom $c\!=\!v$ in σ, $\tau(\Pi)$ contains

$$c\!=\!v \leftarrow Do(c\!=\!v)$$
$$Intervene(c) \leftarrow Do(c\!=\!v).$$

Example 7 continued *The following is $\tau(\Pi)$ for the simple P-log program Π in Example 7 ($x \in \{Mike, John\}$, $b \in \{\mathbf{t}, \mathbf{f}\}$):*

$$Owner(D_1)\!=\!Mike$$
$$Owner(D_2)\!=\!John$$
$$Even(d) \leftarrow Roll(d)\!=\!y,\ y\ mod\ 2 = 0$$
$$\sim\!Even(d) \leftarrow not\ Even(d)$$
$$Roll(d)\!=\!1; Roll(d)\!=\!2; Roll(d)\!=\!3; Roll(d)\!=\!4;$$
$$Roll(d)\!=\!5; Roll(d)\!=\!6 \leftarrow not\ Intervene(Roll(d))$$
$$\leftarrow Obs(Owner(d)\!=\!x), not\ Owner(d)\!=\!x$$
$$\leftarrow Obs(Even(d)\!=\!b), not\ Even(d)\!=\!b$$
$$\leftarrow Obs(Roll(d)\!=\!y), not\ Roll(d)\!=\!y$$

$$Owner(d) = x \leftarrow Do(Owner(d) = x)$$
$$Even(d) = b \leftarrow Do(Even(d) = b)$$
$$Roll(d) = y \leftarrow Do(Roll(d) = y)$$
$$Intervene(Owner(d)) \leftarrow Do(Owner(d) = x)$$
$$Intervene(Even(d)) \leftarrow Do(Even(d) = b)$$
$$Intervene(Roll(d)) \leftarrow Do(Roll(d) = y).$$

The stable models of $\tau(\Pi)$ are called the *possible worlds* of Π, and denoted by $\omega(\Pi)$. For an interpretation W and an atom $c = v$, we say $c = v$ is *possible* in W with respect to Π if Π contains a random selection rule for c

$$[r] \;\; random(c) \leftarrow B,$$

where B is a set of atoms possibly preceded with *not*, and W satisfies B. We say r is *applied* in W if $W \models B$.

We say that a pr-atom $pr_r(c = v \mid B) = p$ is *applied* in W if $W \models B$ and r is applied in W.

As in Baral *et al.* (2009), we assume that simple P-log programs Π satisfy the following conditions:

- **Unique random selection rule** For any constant c, program Π contains at most one random selection rule for c that is applied in W.

- **Unique probability assignment** If Π contains a random selection rule r for constant c that is applied in W, then, for any two different probability atoms

  $$pr_r(c = v_1 \mid B') = p_1$$
  $$pr_r(c = v_2 \mid B'') = p_2$$

  in Π that are applied in W, we have $v_1 \neq v_2$ and $B' = B''$.

Given a simple P-log program Π, a possible world $W \in \omega(\Pi)$ and a constant c for which $c{=}v$ is possible in W, we first define the following notations:

- Since $c{=}v$ is possible in W, by the unique random selection rule assumption, it follows that there is exactly one random selection rule r for constant c that is applied in W. Let $r_{W,c}$ denote this random selection rule. By the unique probability assignment assumption, if there are pr-atoms of the form $pr_{r_{W,c}}(c{=}v \mid B)$ that are applied in W, all B in those pr-atoms should be the same. We denote this B by $B_{W,c}$. Define $PR_W(c)$ as

$$\{pr_{r_{W,c}}(c{=}v \mid B_{W,c}) = p \in \Pi \mid v \in Dom(c)\}.$$

  if $W \not\models Intervene(c)$ and $\emptyset$ otherwise.

- Define $AV_W(c)$ as

$$\left\{v \mid pr_{r_{W,c}}(c{=}v \mid B_{W,c}) = p \in PR_W(c)\right\}.$$

- For each $v \in AV_W(c)$, define the *assigned probability* of $c{=}v$ w.r.t. W, denoted by $ap_W(c{=}v)$, as the value p for which $pr_{r_{W,c}}(c{=}v \mid B_{W,c}) = p \in PR_W(c)$.

- Define the *default probability* for c w.r.t. W, denoted by $dp_W(c)$, as

$$dp_W(c) = \frac{1 - \sum_{v \in AV_W(c)} ap_W(c{=}v)}{|Dom(c) \setminus AV_W(c)|}.$$

For every possible world $W \in \omega(\Pi)$ and every atom $c{=}v$ possible in W, the causal probability $P(W, c{=}v)$ is defined as follows:

$$P(W, c{=}v) = \begin{cases} ap_W(c{=}v) & \text{if } v \in AV_W(c) \\ dp_W(c) & \text{otherwise.} \end{cases}$$

The *unnormalized probability* of a possible world W, denoted by $\hat{\mu}_\Pi(W)$, is defined as

$$\hat{\mu}_\Pi(W) = \prod_{\substack{c=v\in W \text{ and} \\ c=v \text{ is possible in } W}} P(W, c\!=\!v).$$

Assuming Π has at least one possible world with nonzero unnormalized probability, the *normalized probability* of W, denoted by $\mu_\Pi(W)$, is defined as

$$\mu_\Pi(W) = \frac{\hat{\mu}_\Pi(W)}{\sum_{W_i\in\omega(\Pi)} \hat{\mu}_\Pi(W_i)}.$$

Given a simple P-log program Π and a formula A, the probability of A with respect to Π is defined as

$$P_\Pi(A) = \sum_{W \text{ is a possible world of } \Pi \text{ that satisfies } A} \mu_\Pi(W).$$

We say Π is *consistent* if Π has at least one possible world.

Example 7 continued *Given the possible world* $W = \{Owner(D_1)\!=\!Mike, Owner(D_2)\!=\!John, Roll(D_1) = 6,\ Roll(D_2) = 3, Even(D_1)\}$, *the probability of* $Roll(D_1) = 6$ *is* $P(W, Roll(D_1) = 6) = 0.25$, *the probability of* $Roll(D_2) = 3$ *is* $\frac{1}{6}$. *The unnormalized probability of* W, *i.e.,* $\hat{\mu}(W) = P(W, Roll(D_1)\!=\!6) \cdot P(W, Roll(D_2)\!=\!3) = \frac{1}{24}$.

The main differences between simple P-log and P-log are as follows.

- The unique probability assignment assumption in P-log is more general: it does not require the part $B' = B''$. However, all the examples in Baral *et al.* (2009) satisfy our stronger unique probability assignment assumption.

- P-log allows a more general random selection rule of the form

$$[r]\, random(c : \{x : P(x)\}) \leftarrow B'.$$

  Among the examples in Baral *et al.* (2009), only the "Monty Hall Problem" encoding and the "Moving Robot Problem" encoding use "dynamic range $\{x :$

Example	**Parameter**	**plog1**	**plog2**	**Alchemy** $(default)$	**Alchemy** $(maxstep = 5000)$
dice	$N_{dice} = 2$	$0.00s + 0.00s$[a]	$0.00s + 0.00s$[b]	$0.02s + 0.21s$[c]	$0.02s + 0.96s$
	$N_{dice} = 7$	$1.93s + 31.37s$	$0.00s + 1.24s$	$0.13s + 0.73s$	$0.12s + 3.39s$
	$N_{dice} = 8$	$12.66s + 223.02s$	$0.00s + 6.41s$	$0.16s + 0.84s$	$0.16s + 3.86s$
	$N_{dice} = 9$	timeout	$0.00s + 48.62s$	$0.19s + 0.95s$	$0.19s + 4.37s$
	$N_{dice} = 10$	timeout	timeout	$0.23s + 1.06s$	$0.24s + 4.88s$
	$N_{dice} = 100$	timeout	timeout	$19.64s + 16.34s$	$19.55s + 76.18s$
robot	$maxstep = 5$	$0.00s + 0.00s$	segment fault	$2.34s + 2.54s$	$2.3s + 11.75s$
	$maxstep = 10$	$0.37s + 4.86s$	segment fault	$4.78s + 5.24s$	$4.74s + 24.34s$
	$maxstep = 12$	$3.65 + 51.76s$	segment fault	$5.72s + 6.34s$	$5.75s + 29.46s$
	$maxstep = 13$	$11.68s + 168.15s$	segment fault	$6.2s + 6.89s$	$6.2s + 31.96s$
	$maxstep = 15$	timeout	segment fault	$7.18s + 7.99s$	$7.34s + 37.67s$
	$maxstep = 20$	timeout	segment fault	$9.68s + 10.78s$	$9.74s + 50.04s$

Table 4.1: Performance Comparison between Two Ways to Compute Simple P-log Programs

[a] smodels answer set finding time + probability computing time
[b] partial grounding time + probability computing time
[c] mrf creating time + sampling time

$P(x)\}$" in random selection rules and cannot be represented as simple P-log programs.

4.5.2 Turning Simple P-log into Multi-Valued Probabilistic Programs

The main idea of the syntactic translation is to introduce auxiliary probabilistic constants for encoding the assigned probability and the default probability.

Given a simple P-log program Π, a constant c, a set of literals B,[7] and a random selection rule $[r]\, random(c) \leftarrow B'$ in Π, we first introduce several notations, which

[7] A literal is either an atom A or its negation $not\ A$.

resemble the ones used for defining the P-log semantics.

- We define $PR_{B,r}(c)$ as

$$\{pr_r(c{=}v \mid B) = p \in \Pi \mid v \in Dom(c)\}$$

  if Act in Π does not contain $Do(c{=}v')$ for any $v' \in Dom(c)$ and $\emptyset$ otherwise.

- We define $AV_{B,r}(c)$ as

$$\{v \mid pr_r(c{=}v \mid B) = p \in PR_{B,r}(c)\}\,.$$

- For each $v \in AV_{B,r}(c)$, we define the *assigned probability* of $c = v$ w.r.t. B, r, denoted by $ap_{B,r}(c{=}v)$, as the value p for which $pr_r(c{=}v \mid B) = p \in PR_{B,r}(c)$.

- We define the *default probability* for c w.r.t. B and r, denoted by $dp_{B,r}(c)$, as

$$dp_{B,r}(c) = \frac{1 - \sum_{v \in AV_{B,r}(c)} ap_{B,r}(c{=}v)}{|Dom(c) \setminus AV_{B,r}(c)|}.$$

- For each $c \in v$, define its *causal probability* w.r.t. B and r, denoted by $P(B, r, c{=}v)$, as

$$P(B, r, c{=}v) = \begin{cases} ap_{B,r}(c{=}v) & \text{if } v \in AV_{B,r}(c) \\ dp_{B,r}(c) & \text{otherwise.} \end{cases}$$

Now we translate Π into the corresponding multi-valued probabilistic program $\Pi^{\mathrm{LP}^{\mathrm{MLN}}}$ as follows:

- The signature of $\Pi^{\mathrm{LP}^{\mathrm{MLN}}}$ is

$$\begin{aligned} \sigma' &\cup \{pf^c_{B,r}{=}v \mid PR_{B,r}(c) \neq \emptyset \text{ and } v \in Dom(c)\} \\ &\cup \{pf^c_{\Box,r}{=}v \mid r \text{ is a random selection rule of } \Pi \text{ for } c \\ &\qquad\qquad \text{and } v \in Dom(c)\} \\ &\cup \{Assigned_r{=}\mathbf{t} \mid r \text{ is a random selection rule of } \Pi\}. \end{aligned}$$

- $\Pi^{\text{LP}^{\text{MLN}}}$ contains all rules in $\tau(\Pi)$.

- For any constant c, any random selection rule r for c, and any set B of literals such that $PR_{B,r}(c) \neq \emptyset$, include in $\Pi^{\text{LP}^{\text{MLN}}}$:

  – the probabilistic constant declaration:

  $$P(B, r, c = v_1) : pf^c_{B,r} = v_1 \mid \dots$$
  $$\mid P(B, r, c = v_n) : pf^c_{B,r} = v_n$$

  for each probabilistic constant $pf^c_{B,r}$ of the signature, where $\{v_1, \dots, v_n\} = Dom(c)$. The constant $pf^c_{B,r}$ is used for representing the probability distribution for c when condition B holds in the experiment represented by r.

  – the rules

  $$c = v \leftarrow B, B', pf^c_{B,r} = v, not\ Intervene(c). \tag{4.11}$$

  for all $v \in Dom(c)$, where B' is the body of the random selection rule r. These rules assign v to c when the assigned probability distribution applies to $c = v$.

  – the rule

  $$Assigned_r \leftarrow B, B', not\ Intervene(c)$$

  where B' is the body of the random selection rule r (we abbreviate $Assigned_r = \mathbf{t}$ as $Assigned_r$). $Assigned_r$ becomes true when any pr-atoms for c related to r is applied.

- For any constant c and any random selection rule r for c, include in $\Pi^{\text{LP}^{\text{MLN}}}$:

  – the probabilistic constant declaration

  $$\frac{1}{|Dom(c)|} : pf^c_{\Box,r} = v_1 \mid \dots \mid \frac{1}{|Dom(c)|} : pf^c_{\Box,r} = v_n$$

for each probabilistic constant $pf^{c}_{\Box,r}$ of the signature, where $\{v_1,\dots,v_n\} = Dom(c)$. The constant $pf^{c}_{\Box,r}$ is used for representing the default probability distribution for c when there is no applicable pr-atom.

– the rules

$$c{=}v \leftarrow B', pf^{c}_{\Box,r}{=}v, not\ \ Assigned_r.$$

for all $v \in Dom(c)$, where B' is the body of the random selection rule r. These rules assign v to c when the uniform distribution applies to $c{=}v$.

Example 7 continued *The simple P-log program* Π *in Example 7 can be turned into the following multi-valued probabilistic program. In addition to* $\tau(\Pi)$ *we have*

$$\begin{array}{c}
0.25 : pf^{Roll(d)}_{O(d)=M,r(d)}{=}6 \mid 0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}{=}5 \mid \\
0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}{=}4 \mid 0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}{=}3 \mid \\
0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}{=}2 \mid 0.15 : pf^{Roll(d)}_{O(d)=M,r(d)}{=}1
\end{array}$$

$$\begin{array}{c}
\frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}{=}6 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}{=}5 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}{=}4 \mid \\
\frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}{=}3 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}{=}2 \mid \frac{1}{6} : pf^{Roll(d)}_{\Box,r(d)}{=}1
\end{array}$$

$$\begin{array}{r}
Roll(d){=}x \leftarrow Owner(d){=}Mike, pf^{Roll(d)}_{O(d)=M,r(d)}{=}x, \\
not\ \ Intervene(Roll(d))
\end{array}$$

$$Assigned_{r(d)} \leftarrow Owner(d){=}Mike, not\ \ Intervene(Roll(d))$$

$$Roll(d){=}x \leftarrow pf^{Roll(d)}_{\Box,r(d)}{=}x, not\ \ Assigned_{r(d)}.$$

Theorem 12. *For any consistent simple P-log program* Π *of signature* σ *and any possible world* W *of* Π*, we construct a formula* F_W *as follows.*

$$\begin{array}{rl}
F_W = & (\bigwedge_{c=v\in W} c{=}v)\wedge \\
& (\bigwedge_{\substack{c,v:\\ c=v \text{ is possible in } W,\\ W\models c=v \text{ and } PR_W(c)\neq\emptyset}} pf^{c}_{B_{W,c},r_{W,c}}{=}v) \\
& \wedge(\bigwedge_{\substack{c,v:\\ c=v \text{ is possible in } W,\\ W\models c=v \text{ and } PR_W(c)=\emptyset}} pf^{c}_{\Box,r_{W,c}}{=}v)
\end{array}$$

We have

$$\mu_{\Pi}(W) = P_{\Pi^{\mathrm{LP^{MLN}}}}(F_W),$$

and, for any proposition A of signature σ,

$$P_{\Pi}(A) = P_{\Pi^{\mathrm{LP^{MLN}}}}(A).$$

Example 7 continued *For the possible world*

$$\begin{aligned} W = \quad & \{Roll(D_1)\!=\!6, Roll(D_2)\!=\!3, Even(D_1), {\sim}Even(D_2), \\ & Owner(D_1)\!=\!Mike, Owner(D_2)\!=\!John\}, \end{aligned}$$

F_W is

$$\begin{aligned} & Roll(D_1)\!=\!6 \wedge Roll(D_2)\!=\!3 \wedge Even(D_1) \wedge {\sim}Even(D_2) \\ & \wedge\ Owner(D_1)\!=\!Mike \wedge Owner(D_2)\!=\!John \\ & \wedge\ pf^{Roll(D_1)}_{O(D_1)=M,r}\!=\!6 \wedge pf^{Roll(D_2)}_{\Box,r}\!=\!3. \end{aligned}$$

It can be seen that $\hat{\mu}_{\Pi}(W) = \frac{1}{4} \times \frac{1}{6} = P_{\Pi^{\mathrm{LP^{MLN}}}}(F_W)$.

The embedding tells us that the exact inference in simple P-log is no harder than the one in $\mathrm{LP^{MLN}}$.

4.6 Other Related Work

Sato's distribution semantics (Sato (1995)) defines probability distributions over truth assignments on a set of independent "choice atoms" , which further derive fully specified possible worlds through logic programs. Problog (De Raedt and Kimmig (2015)) is one formalism with the distribution semantics. Other examples include PRISM (Sato and Kameya (1997)), Poole's Independent Choice Logic (ICL; Poole (1997)) Logic Programs with Annotated Disjunction (LPAD; Vennekens *et al.* (2004)), etc. They are defined with differences in the syntax and semantics of their logic component. PRISM requires the rules in the logic program to be definite clauses; Poole's ICL allows arbitrary acyclic logic programs; LPAD has the choice atoms introduced together with rules in the logic programs by associating an annotation with each disjunctive term, specifying the probability of the disjunctive term; ProbLog adopts definite clauses with well-founded semantics[8].

The result that ProbLog can be viewed as a special case of LP$^{\text{MLN}}$ can be extended to embed Logic Programs with Annotated Disjunctions (LPAD) in LP$^{\text{MLN}}$ based on the fact that any LPAD program can be further turned into a ProbLog program by eliminating disjunctions in the heads (Gutmann, 2011, Section 3.3).

It is known that LPAD is related to several other languages. In Vennekens *et al.* (2004), it is shown that Poole's ICL (Poole (1997)) can be viewed as LPAD, and that acyclic LPAD programs can be turned into ICL. This indirectly tells us how ICL is related to LP$^{\text{MLN}}$.

CP-logic (Vennekens *et al.* (2009)) 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.

PrASP (Nickles and Mileo (2014)) is a recent language similar to LP$^{\text{MLN}}$ in that the probability distribution is obtained from the annotations of the formulas. However, in PrASP, the annotations of formulas are explicitly probabilities of the annotated formulas, whereas in LP$^{\text{MLN}}$ the probabilities of rules need to be derived from the weights in a computationally expensive way. An LP$^{\text{MLN}}$ program specify exactly one probability distribution over stable models, while in PrASP, there can be none or multiple probability distributions satisfying the marginal probabilities of formulas specified by a program. There are also other probabilistic extensions of stable model semantics such as Ng and Subrahmanian (1994) and Saad and Pontelli (2005).

Similar to LP$^{\text{MLN}}$, log-linear description logics (Niepert *et al.* (2011)) follow the weight scheme of log-linear models in the context of description logics.

[8]An extension of ProbLog, called cProbLogFierens *et al.* (2012), allows constraints.

4.7 Proofs

4.7.1 Proof of Theorem 3

Theorem 3 *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.*

Proof. We notice that $\overline{(\mathbb{P}_\Pi)^{hard}} = \Pi$. We first show that an interpretation I is a stable model of Π if and only if it is a stable model of $\mathbb{P}_\Pi$ whose weight is $e^{k\alpha}$. Suppose I is a stable model of Π. Then I is a stable model of $\overline{(\mathbb{P}_\Pi)^{hard}}$. Obviously $(\mathbb{P}_\Pi)^{hard}$ is $(\mathbb{P}_\Pi)_I$. So the weight of I is $e^{k\alpha}$. Suppose I is a stable model of $\mathbb{P}_\Pi$ whose weight is $e^{k\alpha}$. Then I satisfies all the rules in $\mathbb{P}_\Pi$, since all rules in $\mathbb{P}_\Pi$ contribute to its weight, and I is a stable model of $\overline{(\mathbb{P}_\Pi)_I} = \overline{(\mathbb{P}_\Pi)^{hard}}$, which is equivalent to Π. So I is a stable model of Π.

Now suppose Π has at least one stable model. It follows that $\overline{(\mathbb{P}_\Pi)^{hard}}$ has some stable model.

- Suppose I is not a stable model of Π.
  - Suppose I does not satisfy Π. Then $I \not\models \overline{(\mathbb{P}_\Pi)^{hard}}$. By Proposition 3, $P_{\mathbb{P}_\Pi}(I) = 0$, and consequently I is not a stable model of $\mathbb{P}_\Pi$.
  - Suppose I satisfies Π. Then $\overline{(\mathbb{P}_\Pi)_I} = \Pi$ and I is not a stable model of $\overline{(\mathbb{P}_\Pi)_I}$. By definition, $W_{\mathbb{P}_\Pi}(I) = 0$ and consequently $P_{\mathbb{P}_\Pi}(I) = 0$, which means that I is not a stable model of $\mathbb{P}_\Pi$.

- Suppose I is a stable model of Π. Then $I \models \overline{(\mathbb{P}_\Pi)^{hard}}$, $\overline{(\mathbb{P}_\Pi)_I} = \Pi$ and I is a stable model of $\overline{(\mathbb{P}_\Pi)_I}$.

  By Proposition 3,

$$P_{\mathbb{P}_\Pi}(I) = \frac{exp(\sum_{w:F \in (\mathbb{P}_\Pi)_I \setminus (\mathbb{P}_\Pi)^{hard}} w)}{\sum_{J \models_{SM} \overline{(\mathbb{P}_\Pi)_J} : J \models \overline{(\mathbb{P}_\Pi)^{hard}}} exp(\sum_{w:F \in \overline{(\mathbb{P}_\Pi)_J \setminus (\mathbb{P}_\Pi)^{hard}}} w)}$$
$$= \frac{exp(0)}{\sum_{J \models_{SM} \overline{(\mathbb{P}_\Pi)_J} : J \models \overline{(\mathbb{P}_\Pi)^{hard}}} exp(\sum_{w:F \in \overline{(\mathbb{P}_\Pi)_J \setminus (\mathbb{P}_\Pi)^{hard}}} w)}.$$

  It can be seen that "$J \models_{SM} \overline{(\mathbb{P}_\Pi)_J} : J \models \overline{(\mathbb{P}_\Pi)^{hard}}$" is equivalent to "$J$ is a stable model of Π", since $\overline{(\mathbb{P}_\Pi)^{hard}} = \Pi$. Furthermore, since $\Pi \setminus \overline{(\mathbb{P}_\Pi)^{hard}} = \emptyset$, we have $exp(\sum_{w:F \in \overline{(\mathbb{P}_\Pi)_J \setminus (\mathbb{P}_\Pi)^{hard}}} w) = exp(0)$ for all $J \models_{SM} \overline{\mathbb{P}_J} : J \models \overline{(\mathbb{P}_\Pi)^{hard}}$. So

$$P_{\mathbb{P}_\Pi}(I) = \frac{exp(0)}{\sum_{J \models_{SM} \Pi} exp(0)}$$
$$= \frac{1}{k}$$

  where k is the number of stable models of Π.

$\square$

4.7.2 Proof of Proposition 6

Proof. Firstly, it is easy to see that the second and third conditions are equivalent. We notice that $exp(x) > 0$ for all $x \in (-\infty, +\infty)$. So it can be seen from the definition that $W'_\Pi(I) > 0$ if and only if $I \in SM'[\Pi]$, and consequently $P'_\Pi(I) > 0$ if and only if $I \in SM'[\Pi]$.

Secondly, by Proposition 3, we know that $P'_\Pi(I)$ is equivalent to $P_\Pi(I)$. By definition, the first condition is equivalent to "$P_\Pi(I) > 0$". So we have that the first condition is equivalent to the third condition. $\square$

Proposition 5 does not hold if we replace "$SM'[\Pi]$" by "$SM[\Pi]$".

Example 8. *Consider the following* LP$^{\text{MLN}}$ *program* Π*:*

$$\begin{array}{lll} (r_1) & \alpha: & p \\ (r_2) & 1: & q \end{array}$$

and the interpretation $I = \{q\}$*.* I *belongs to* $SM[\Pi]$ *since* I *is a stable model of* $\overline{\Pi_I}$*, which contains* r_2 *only. However,* $P_\Pi(I) = 0$ *since* I *does not satisfy the hard rule* r_1*. On the other hand,* I *does not belong to* $SM'[\Pi]$*.*

To facilitate the proof of Proposition 6, we introduce a formal definition of ASP programs with weak constraints, as follows.

An *ASP program with weak constraints* is a pair

$$\langle \Pi, \mathit{CONSTR} \rangle,$$

where Π is a set of standard ASP rules of the form (2.1), and CONSTR is a set of weak constraints C of the following form

$$:\sim \mathit{Body} \quad [\mathit{Weight}], \tag{4.12}$$

where $Weight$ is a positive integer, and *Body* is a set of literals. We will refer to *Body* by $\mathit{Body}(C)$, and *Weight* by $Weight(C)$. The *penalty* that I receives, denoted as $Penalty(I)$, is defined as

$$Penalty(I) = \sum_{C \in \mathit{CONSTR}: I \models \mathit{Body}(C)} Weight(C).$$

The *stable models* of an ASP program with weak constraints $\langle \Pi, \mathit{CONSTR} \rangle$ are the elements of the following set

$$\begin{aligned} \{I \mid I \models_{\text{SM}} \Pi \text{ and there does not exists } J \neq I \text{ such that} \\ J \models_{\text{SM}} \Pi \text{ and } Penalty(J) < Penalty(I)\}. \end{aligned}$$

By $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$ we denote the following LP$^{\text{MLN}}$ program:

$$\{\alpha : R \mid R \in \Pi\} \cup \{-Weight(C) : \bot \leftarrow \mathit{Body}(C) \mid C \in \mathit{CONSTR}\}\,.$$

For any interpretation K, let CONSTR_K denote the following set:

$$\{\bot \leftarrow \mathit{Body}(C) \mid C \in \mathit{CONSTR}, K \models \neg \mathit{Body}(C)\}$$

Lemma 3. *For any program with weak constraints $\langle \Pi, \mathit{CONSTR} \rangle$ that has a stable model, an interpretation I is a stable model of Π if and only if I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$.*

Proof. ($\Rightarrow$) Since CONSTR_I consists of constraints only, we can derive from the fact that I is a stable model of Π that I is a stable model of $\Pi \cup \mathit{CONSTR}_I$, which is $\overline{(\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}})^{hard}} \cup \overline{(((\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}})^{soft})_I}$. So $I \in \mathit{SM}' \left[\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}\right]$ and by Proposition 5, I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$.

($\Leftarrow$) Consider any stable model I of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$. By Proposition 5, $I \in \mathit{SM}' \left[\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}\right]$. This means I is a stable model of $\overline{(\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}})^{hard}} \cup \overline{(((\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}})^{soft})_I}$, which is equivalent to $\Pi \cup \mathit{CONSTR}_I$. Since CONSTR_I contains constraints only, I is a stable model of Π.

$\square$

Proposition 6 *For any program with weak constraints that has a stable model, its stable models are the same as the stable models of the corresponding* LP$^{\text{MLN}}$ *program with the highest normalized weight.*

Proof. ($\Rightarrow$) For any program with weak constraints $\langle \Pi, \mathit{CONSTR} \rangle$ that has a stable model, let I be any one of its stable models. Since I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle$, by definition, we have:

1. $I \models_{\text{SM}} \Pi$;

2. There does not exist $J \neq I$ such that $J \models_{\text{SM}} \Pi$ and $\mathit{Penalty}(J) < \mathit{Penalty}(I)$.

From the first condition, by Lemma 3, it follows that I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$.

Now we show that there does not exist any $J \neq I$ such that J is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$ and $P_{\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}}(J) > P_{\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}}(I)$. Assume, for the sake of contradiction, that such J exists. Then J must be a stable model of Π by Lemma 3. Since $P_{\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}}(J) > P_{\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}}(I)$, due to how we translate $\langle \Pi, \mathit{CONSTR} \rangle$ to $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$, $\mathit{Penalty}(J) < \mathit{Penalty}(I)$, which is a contradiction to the second condition. So such J does not exist.

So I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$ with the highest normalized weight.

($\Leftarrow$) Let I be any stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$ with the highest normalized weight.

- $I \models_{\text{SM}} \Pi$: Since I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$, by Lemma 3, I is a stable model of Π.

- **There does not exist any J s.t. $J \models_{\text{SM}} \Pi$ and $\mathit{Penalty}(J) < \mathit{Penalty}(I)$:** Suppose, to the contrary, that there exists such J. By Lemma 3, J is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$. Since $\mathit{Penalty}(J) < \mathit{Penalty}(I)$, $P_{\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}}(J) > P_{\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}}(I)$. This is a contradiction to the fact that I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle^{\text{LP}^{\text{MLN}}}$ with the highest normalized weight. So there cannot exist such J.

In conclusion, I is a stable model of $\langle \Pi, \mathit{CONSTR} \rangle$.

$\square$

4.7.3 Proof of Theorem 4 and Theorem 5

We divide the ground program obtained from $\mathsf{lpmln2asp}^{\mathsf{rwd}}(\Pi)$ into three parts:

$$SAT(\Pi) \cup ORIGIN(\Pi) \cup WC(\Pi)$$

where

$$SAT(\Pi) = \{\mathtt{sat}(i, w_i, \mathbf{c}) \leftarrow Head_i(\mathbf{c}) \mid w_i : \ Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \in Gr(\Pi)\} \cup$$
$$\{\mathtt{sat}(i, w_i, \mathbf{c}) \leftarrow \mathtt{not}\ Body_i(\mathbf{c}) \mid w_i : \ Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \in Gr(\Pi)\}$$

$$ORIGIN(\Pi) =$$
$$\{Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}), \mathtt{not\ not\ sat}(i, w_i, \mathbf{c}) \mid w_i : \ Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \in Gr(\Pi)\}$$

and

$$WC(\Pi) = \{:\sim \mathtt{sat}(i, w_i, \mathbf{c}). \ \ [-w_i@l, i, \mathbf{c}] \mid w_i : \ Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \in Gr(\Pi)\}$$

Lemma 4. *For any* LP$^{\text{MLN}}$ *program* Π,

$$\phi(I) = I \cup \{\mathtt{sat}(i, w_i, \mathbf{c}) \mid w_i : Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \in Gr(\Pi), I \models Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c})\}$$

is a 1-1 correspondence between SM[Π] *and the stable models of* $SAT(\Pi) \cup ORIGIN(\Pi)$.

Proof. Let σ be the signature of Π, and let σ_{sat} be the set

$$\{\mathtt{sat}(i, w_i, \mathbf{c}) \mid w_i : \ Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}) \in Gr(\Pi)\}.$$

It can be seen that

- each strongly connected component of the dependency graph of $SAT(\Pi) \cup ORIGIN(\Pi)$ w.r.t. $\sigma \cup \sigma_{sat}$ is a subset of σ or a subset of σ_{sat};
- no atom in σ_{sat} has a strictly positive occurrence in $ORIGIN(\Pi)$;

- no atom in σ has a strictly positive occurrence in $SAT(\Pi)$.

Thus, according to the splitting theorem, $\phi(I)$ is a stable model of $SAT(\Pi) \cup ORIGIN(\Pi)$ if and only if $\phi(I)$ is a stable model of $SAT(\Pi)$ w.r.t. σ_{sat} and is a stable model of $ORIGIN(\Pi)$ w.r.t. σ.

First, assuming that I belongs to SM[Π], we will prove that $\phi(I)$ is a stable model of $SAT(\Pi) \cup ORIGIN(\Pi)$. Let I be a member of SM[Π].

- **$\phi(I)$ is a stable model of $SAT(\Pi)$ w.r.t. σ_{sat}.** By the definition of ϕ, $\mathtt{sat}(i, w_i, \mathbf{c}) \in \phi(I)$ if and only if $I \models Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c})$, in which case either $I \models Head_i(\mathbf{c})$ or $I \not\models Body_i(\mathbf{c})$. This means

  $$\phi(I) \models SAT(\Pi) \cup$$
  $$\{\mathtt{sat}(i, w_i, \mathbf{c}) \rightarrow Head_i(\mathbf{c}) \vee \neg Body_i(\mathbf{c}) \mid w_i : Head_i(\mathbf{c}) \leftarrow \ Body_i(\mathbf{c}) \in Gr(\Pi)\},$$

  which is the completion of $SAT(\Pi)$. It is obvious that $SAT(\Pi)$ is tight on σ_{sat}. So $\phi(I)$ is a stable model of $SAT(\Pi)$ w.r.t. σ_{sat}.

- **$\phi(I)$ is a stable model of $ORIGIN(\Pi)$ w.r.t. σ.** It is clear that $\phi(I)$ satisfies $ORIGIN(\Pi)$. Assume for the sake of contradiction that there is an interpretation $J \subset \phi(I)$ such that J and $\phi(I)$ agree on σ^{sat} and $J \models ORIGIN(\Pi)^{\phi(I)}$. Then

  $$J \models Head_i(\mathbf{c})^{\phi(I)} \leftarrow Body_i(\mathbf{c})^{\phi(I)}, (\mathtt{not\ not\ sat}(i, w_i, \mathbf{c}))^{\phi(I)}$$

  for every rule

  $$Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c}), \mathtt{not\ not\ sat}(i, w_i, \mathbf{c})$$

  in $ORIGIN(\Pi)$. Since $\phi(I)$ satisfies $SAT(\Pi)$, it follows that for every rule $Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c})$ satisfied by $\phi(I)$, we have $(\mathtt{not\ not\ sat}(i, w_i, \mathbf{c}))^{\phi(I)} = \top$ so that $J \models Head_i(\mathbf{c})^{\phi(I)} \leftarrow Body_i(\mathbf{c})^{\phi(I)}$, or equivalently, $J \models Head_i(\mathbf{c})^I \leftarrow Body_i(\mathbf{c})^I$, which contradicts that I is a stable model of $\overline{\Pi_I}$.

Consequently, by the splitting theorem, $\phi(I)$ is a stable model of $SAT(\Pi) \cup ORIGIN(\Pi)$.

Next, assuming $\phi(I)$ is a stable model of $SAT(\Pi) \cup ORIGIN(\Pi)$, we will prove that I belongs to SM[Π].

Let $\phi(I)$ be a stable model of $SAT(\Pi) \cup ORIGIN(\Pi)$. By the splitting theorem, $\phi(I)$ is a stable model of $SAT(\Pi)$ w.r.t. σ_{sat} and $\phi(I)$ is a stable model of $ORIGIN(\Pi)$ w.r.t. σ.

It is clear that $I \models \overline{\Pi_I}$.

Assume for the sake of contradiction that there is an interpretation $J \subset I$ such that $J \models (\overline{\Pi_I})^I$. Take any rule

$$(\mathit{Head}_i(\mathbf{c}))^{\phi(I)} \leftarrow (\mathit{Body}_i(\mathbf{c}))^{\phi(I)}, (\texttt{not not sat}(i, w_i, \mathbf{c}))^{\phi(I)} \tag{4.13}$$

in $(ORIGIN(\Pi))^{\phi(I)}$.

Case 1: $\phi(I) \not\models \texttt{sat}(i, w_i, \mathbf{c})$. Clearly, $J \models (4.13)$.

Case 2: $\phi(I) \models \texttt{sat}(i, w_i, \mathbf{c})$. Since $\mathit{Head}_i(\mathbf{c})$ and $\mathit{Body}_i(\mathbf{c})$ do not contain $\texttt{sat}$ predicates, (4.13) is equivalent to

$$(\mathit{Head}_i(\mathbf{c}))^I \leftarrow (\mathit{Body}_i(\mathbf{c}))^I. \tag{4.14}$$

Since $\phi(I)$ is a stable model of $SAT(\Pi)$ w.r.t. σ_{sat}, we have $\phi(I) \models \mathit{Head}_i(\mathbf{c}) \leftarrow \mathit{Body}_i(\mathbf{c})$, or equivalently, $I \models \mathit{Head}_i(\mathbf{c}) \leftarrow \mathit{Body}_i(\mathbf{c})$. So, $\mathit{Head}_i(\mathbf{c}) \leftarrow \mathit{Body}_i(\mathbf{c}) \in \overline{\Pi_I}$, and $\mathit{Head}_i(\mathbf{c})^I \leftarrow \mathit{Body}_i(\mathbf{c})^I \in (\overline{\Pi_I})^I$. Since $J \models (\overline{\Pi_I})^I$, it follows that $J \models (4.13)$ as well.

Since $J \subset \phi(I)$, $\phi(I)$ is not a stable model of $ORIGIN(\Pi)$ w.r.t. σ, which contradicts the assumption that it is. Thus we conclude that I is a stable model of $\overline{\Pi_I}$, i.e., I belongs to SM[Π]. □

Theorem 4 For any $\mathrm{LP}^{\mathrm{MLN}}$ program Π, there is a 1-1 correspondence ϕ between $\mathrm{SM}[\Pi]$ and the set of stable models of $\mathsf{lpmln2asp}^{\mathsf{rwd}}(\Pi)$, where

$$\phi(I) = I \cup \{\mathtt{sat}(i, w_i, \mathbf{c}) \mid w_i : \mathit{Head}_i(\mathbf{c}) \leftarrow \mathit{Body}_i(\mathbf{c}) \text{ in } \mathit{Gr}(\Pi), I \models \mathit{Body}_i(\mathbf{c}) \rightarrow \mathit{Head}_i(\mathbf{c})\}.$$

Furthermore,

$$W_\Pi(I) = exp\Bigg(\sum_{\mathtt{sat}(i,w_i,\mathbf{c}) \in \phi(I)} w_i\Bigg).$$

Also, ϕ is a 1-1 correspondence between the most probable stable models of Π and the optimal stable models of $\mathsf{lpmln2asp}^{\mathsf{rwd}}(\Pi)$.

Proof. By Lemma 4, ϕ is a 1-1 correspondence between $\mathrm{SM}[\Pi]$ and the set of stable models of $\mathsf{lpmln2asp}^{rwd}(\Pi)$.

The fact

$$W_\Pi(I) = exp\Bigg(\sum_{\mathtt{sat}(i,w_i,\mathbf{c}) \in \phi(I)} w_i\Bigg). \tag{4.15}$$

can be easily seen from how $\phi(I)$ is defined.

It remains to show that ϕ is a 1-1 correspondence between the most probable stable models of Π and the optimal stable models of $\mathsf{lpmln2asp}^{\mathsf{rwd}}(\Pi)$. For any interpretation I of $\mathsf{lpmln2asp}^{\mathsf{rwd}}(\Pi)$, we use $Penalty_\Pi(I, l)$ to denote the total penalty it receives at level l defined by weak constraints:

$$Penalty_\Pi(I, l) = \sum_{\substack{:\sim \mathtt{sat}(i,w_i,\mathbf{c}).[-w_i'@l,i,\mathbf{c}] \in WC(\Pi),\\ I \models \mathtt{sat}(i,w_i,\mathbf{c})}} -w_i$$

Let $\phi(I)$ be a stable model of $\mathsf{lpmln2asp}^{\mathsf{rwd}}(\Pi)$. By Lemma 4, $I \in \mathrm{SM}[\Pi]$. So it is sufficient to prove

$$I \in \underset{J:J\in \underset{K:K\in SM[\Pi]}{argmax} W_{\Pi^{\text{hard}}}(K)}{argmax} W_{\Pi^{\text{soft}}}(J)$$

$$\text{iff} \tag{4.16}$$

$$\phi(I) \in \underset{J':J'\in \underset{K':K' \text{ is a stable model of } \mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}{argmin} Penalty_{\mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}(K',1)}{argmin} Penalty_{\mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}(J',0).$$

This is true because (we abbreviate $Head_i(\mathbf{c}) \leftarrow Body_i(\mathbf{c})$ as $F_i(\mathbf{c})$)

$$I \in \underset{J:\ J\in \underset{K:\ K\in \text{SM}[\Pi]}{\text{argmax}} W_{\Pi^{\text{hard}}}(K)}{\text{argmax}} W_{\Pi^{\text{soft}}}(J)$$

iff (by Lemma 4 and definition)

$$\phi(I) \in \underset{J':\ J'\in \underset{K':\ K' \text{ is a stable model of } \mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}{\text{argmax}} exp\Big(\sum\limits_{\alpha:F_i(\mathbf{c})\ \in\ (\Pi^{\text{hard}})_{K'}} \alpha\Big)}{\text{argmax}} exp\Big(\sum_{w_i:F_i(\mathbf{c})\ \in\ (\Pi^{\text{soft}})_{J'}} w_i\Big)$$

iff

$$\phi(I) \in \underset{J':\ J'\in \underset{K':\ K' \text{ is a stable model of } \mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}{\text{argmax}} exp\Big(\sum\limits_{\alpha:F_i(\mathbf{c})\ \in\ \Pi^{\text{hard}},K'\models F_i(\mathbf{c})} 1\Big)}{\text{argmax}} exp\Big(\sum_{w_i:F_i(\mathbf{c})\ \in\ \Pi^{\text{soft}},J'\models F_i(\mathbf{c})} w_i\Big)$$

iff

$$\phi(I) \in \underset{J':\ J'\in \underset{K':\ K' \text{ is a stable model of } \mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}{\text{argmin}} \Big(\sum\limits_{\alpha:F_i(\mathbf{c})\ \in\ \Pi^{\text{hard}},K'\models F_i(\mathbf{c})} -1\Big)}{\text{argmin}} \Big(\sum_{w_i:F_i(\mathbf{c})\ \in\ \Pi^{\text{soft}},J'\models F_i(\mathbf{c})} -w_i\Big)$$

iff

$$\phi(I) \in \underset{J':\ J'\in \underset{K':\ K' \text{ is a stable model of } \mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}{\text{argmin}} \Bigg(\sum\limits_{\substack{:\sim \mathtt{sat}(i,w_i,\mathbf{c}).[-1@1,i,\mathbf{c}] \\ \in\ \mathsf{lpmln2asp}^{\text{rwd}}(\Pi), \\ K'\models \mathtt{sat}(i,w_i,\mathbf{c})}} -1\Bigg)}{\text{argmin}} \Bigg(\sum_{\substack{:\sim \mathtt{sat}(i,w_i,\mathbf{c}).[-w_i@0,i,\mathbf{c}] \\ \in \mathsf{lpmln2asp}^{\text{rwd}}(\Pi), \\ J'\models \mathtt{sat}(i,w_i,\mathbf{c})}} -w_i\Bigg)$$

iff

$$\phi(I) \in \underset{J':\ J'\in \underset{K':\ K' \text{ is a stable model of } \mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}{\text{argmin}} Penalty_{\mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}(K',1)}{\text{argmin}} Penalty_{\mathsf{lpmln2asp}^{\text{rwd}}(\Pi)}(J',0).$$

$\square$

Theorem 5 can be proven similarly to Theorem 4.

4.7.4 Proof of Theorem 6 and Theorem 8

Given a signature σ, we use $\boldsymbol{At}(\sigma)$ to denote the set of all ground atoms that can be constructed from symbols in σ.

Theorem 6 *Any MLN* $\mathbb{L}$ *and its* LP$^{\text{MLN}}$ *representation* $\Pi_{\mathbb{L}}$ *have the same probability distribution over all interpretations.*

Proof. We show that for any interpretation I, $P_{\mathbb{L}}(I) = P_{\Pi_{\mathbb{L}}}(I)$. For a set of atoms $\mathbf{p}$, let $\mathbf{Choice}(\mathbf{p})$ denote the set of weighted rules $\bigcup_{p\in\mathbf{p}} \{w : p \leftarrow \textit{not not } p\}$.

$$\begin{aligned}
P_{\mathbb{L}}(I) &= \lim_{\alpha\to\infty} \frac{W_{\mathbb{L}}(I)}{\sum_{J\in PW} W_{\mathbb{L}}(J)} \\
&= \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\mathbb{L}_I} w)}{\sum_{J\in PW} exp(\sum_{w:F\in\mathbb{L}_J} w)}.
\end{aligned}$$

Multiplying the weight of every interpretation by $exp(|\boldsymbol{At}(\sigma)|\cdot w)$, we have

$$\begin{aligned}
P_{\mathbb{L}}(I) &= \lim_{\alpha\to\infty} \frac{exp(|\boldsymbol{At}(\sigma)|\cdot w)\cdot exp(\sum_{w:F\in\mathbb{L}_I} w)}{\sum_{J\in PW} exp(|\boldsymbol{At}(\sigma)|\cdot w)\cdot exp(\sum_{w:F\in\mathbb{L}_J} w)} \\
&= \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\mathbb{L}_I\cup\mathbf{Choice}(\boldsymbol{At}(\sigma))} w)}{\sum_{J\in PW} exp(\sum_{w:F\in\mathbb{L}_J\cup\mathbf{Choice}(\boldsymbol{At}(\sigma))} w)}.
\end{aligned}$$

Clearly $\mathbf{Choice}(\boldsymbol{At}(\sigma))$ is a set of tautologies, and it can be seen from the construction of $\Pi_{\mathbb{L}}$ that $(\Pi_{\mathbb{L}})_K = \mathbb{L}_K \cup \mathbf{Choice}(\boldsymbol{At}(\sigma))$ for any interpretation K. So

$$P_{\mathbb{L}}(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in(\Pi_{\mathbb{L}})_I} w)}{\sum_{J\in PW} exp(\sum_{w:F\in(\Pi_{\mathbb{L}})_J} w)}.$$

By Theorem 2 in Ferraris *et al.* (2011), for any interpretation K, the stable models of $\overline{(\Pi_{\mathbb{L}})_K}$ are exactly the models of $\overline{\mathbb{L}_K}$. Since K itself is a model of $\overline{\mathbb{L}_K}$, K is a stable model of $\overline{(\Pi_{\mathbb{L}})_K}$. So

$$
\begin{aligned}
P_{\mathbb{L}}(I) &= \lim_{\alpha\to\infty} \frac{W_{\Pi_{\mathbb{L}}}(I)}{\sum_{J\in SM[\Pi]} W_{\Pi_{\mathbb{L}}}(J)} \\
&= P_{\Pi_{\mathbb{L}}}(I).
\end{aligned}
$$

□

For a (deterministic) logic program Π, we use LF_{Π} to denote the set

$$\{LF_{\Pi}(L) \mid L \text{ is a loop of } \Pi\}$$

.

Lemma 5. *For any* LP$^{\text{MLN}}$ *program* Π *and any interpretation* I *of the underlying signature* σ*,* $I \models LF_{\overline{\Pi}}$ *if and only if* $I \models LF_{\overline{\Pi_I}}$*.*

Proof. ($\Rightarrow$) Suppose $I \models LF_{\overline{\Pi}}$. Consider any subset K of σ. There are two possible cases:

- $I \not\models K^{\wedge}$. In this case, $K^{\wedge} \rightarrow ES_{\overline{\Pi_I}}(K)$ is trivially satisfied by I.
- $I \models K^{\wedge}$. Since $I \models LF_{\overline{\Pi}}$, by Theorem 2, we have

  $$K^{\wedge} \rightarrow \bigvee_{\substack{A\cap K\neq\emptyset\\ A\leftarrow B\wedge N\in\overline{\Pi}\\ B\cap K=\emptyset}} (B \wedge N \wedge \bigwedge_{b\in A\setminus K} \neg b)$$

  is satisfied by I. Consider the rules which contribute to the external support for K in $\overline{\Pi}$, i.e., $A \leftarrow B \wedge N \in \overline{\Pi}$ such that $A \cap K \neq \emptyset$ and $B \cap K = \emptyset$. Since

$A \cap K \neq \emptyset$ and $I \models K^{\wedge}$, we get $I \models A^{\vee}$. So all these rules are satisfied by I and thus they all belong to $\overline{\Pi_I}$, which means

$$\bigvee_{\substack{A\cap K\neq\emptyset\\ A\leftarrow B\wedge N\in\overline{\Pi}\\ B\cap K=\emptyset}} (B \wedge N \wedge \bigwedge_{b\in A\setminus K} \neg b) = \bigvee_{\substack{A\cap K\neq\emptyset\\ A\leftarrow B\wedge N\in\overline{\Pi_I}\\ B\cap K=\emptyset}} (B \wedge N \wedge \bigwedge_{b\in A\setminus K} \neg b).$$

So

$$K^{\wedge} \rightarrow \bigvee_{\substack{A\cap K\neq\emptyset\\ A\leftarrow B\wedge N\in\overline{\Pi_I}\\ B\cap K=\emptyset}} (B \wedge N \wedge \bigwedge_{b\in A\setminus K} \neg b)$$

i.e.,

$$K^{\wedge} \rightarrow ES_{\overline{\Pi_I}}(K)$$

is satisfied by I.

In conclusion, I satisfies $K^{\wedge} \rightarrow ES_{\overline{\Pi_I}}(K)$ for all subsets K of σ. By Theorem 2, $I \models LF_{\overline{\Pi_I}}$.

($\Leftarrow$) (The reasoning is similar to the proof of Proposition 1) Suppose I satisfies $LF_{\overline{\Pi_I}}$. For all subsets L of σ, since $I \models LF_{\overline{\Pi_I}}$, by Theorem 2, $I \models L^{\wedge} \rightarrow ES_{\overline{\Pi_I}}(L)$. Since $\Pi_I \subseteq \Pi$, it can be seen that the disjunctive terms in $ES_{\overline{\Pi_I}}(L)$ is a subset of the disjunctive terms in $ES_{\overline{\Pi}}(L)$, and thus $ES_{\overline{\Pi_I}}(L)$ entails $ES_{\overline{\Pi}}(L)$. So $I \models L^{\wedge} \rightarrow ES_{\overline{\Pi}}(L)$. So $I \models LF_{\overline{\Pi}}$. $\square$

Lemma 6. *Let $\mathbb{L}$ be an MLN, and let $\mathbb{L}^{hard}$ be the hard formulas in $\mathbb{L}$. Let $\overline{\mathbb{L}^{hard}}$ be the set of formulas obtained from $\mathbb{L}^{hard}$ by dropping all weights. When $\overline{\mathbb{L}^{hard}}$ is satisfiable,*

- *if I satisfies $\overline{\mathbb{L}^{hard}}$,*

$$P_{\mathbb{L}}(I) = \frac{exp(\sum_{w:F\in\mathbb{L}_I\setminus\mathbb{L}^{hard}} w)}{\sum_{J\in PW:J\models\overline{\mathbb{L}^{hard}}} exp(\sum_{w:F\ \in\ \mathbb{L}_J\setminus\mathbb{L}^{hard}} w)}$$

- *otherwise,* $P_{\mathbb{L}}(I) = 0$.[9]

Proof. For any interpretation I, by definition, we have

$$\begin{aligned} P_{\mathbb{L}}(I) &= \lim_{\alpha\to\infty} \frac{W_{\mathbb{L}}(I)}{\sum_{J\in PW} W_{\mathbb{L}}(J)} \\ &= \lim_{\alpha\to\infty} \frac{W_{\mathbb{L}}(I)}{\sum_{J\in PW} exp(\sum_{w:F\in\mathbb{L}_J} w)}. \end{aligned}$$

- Suppose I satisfies $\overline{\mathbb{L}^{hard}}$. We have

$$P_{\mathbb{L}}(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\mathbb{L}_I} w)}{\sum_{J\in PW} exp(\sum_{w:F\in\mathbb{L}_J} w)}.$$

  Splitting the denominator into two parts: those J that satisfy $\overline{\mathbb{L}^{hard}}$ and those that do not, and extracting the weight of formulas in $\mathbb{L}^{hard}$, we have

$$P_{\mathbb{L}}(I) = \lim_{\alpha\to\infty} \frac{exp(|\mathbb{L}^{hard}|\cdot\alpha)\cdot exp(\sum_{w:F\in\mathbb{L}_I\setminus\mathbb{L}^{hard}} w)}{HSAT + HUNSAT}.$$

  where

$$HSAT = exp(|\mathbb{L}^{hard}|\cdot\alpha)\cdot \sum_{J\models\overline{\mathbb{L}^{hard}}} exp(\sum_{w:F\in\mathbb{L}_J\setminus\mathbb{L}^{hard}} w)$$

  are weights from those J's that satisfy $\overline{\mathbb{L}^{hard}}$, and

$$HUNSAT = \sum_{J\not\models\overline{\mathbb{L}^{hard}}} exp(|\mathbb{L}^{hard}\cap\mathbb{L}_J|\cdot\alpha)\cdot exp(\sum_{w:F\in\mathbb{L}_J\setminus\mathbb{L}^{hard}} w)$$

  are weights from those J's that do not satisfy $\overline{\mathbb{L}^{hard}}$.

  We divide both the numerator and the denominator by $exp(|\mathbb{L}^{hard}|\cdot\alpha)$.

$$P_{\mathbb{L}}(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{w:F\in\mathbb{L}_I\setminus\mathbb{L}^{hard}} w)}{\frac{HSAT}{exp(|\mathbb{L}^{hard}|\cdot\alpha)} + \frac{HUNSAT}{exp(|\mathbb{L}^{hard}|\cdot\alpha)}}$$

[9] This proposition does not hold when $\overline{\mathbb{L}^{hard}}$ is not satisfiable. For example, consider $\mathbb{L} = \{\alpha : p, \alpha :\leftarrow p\}$ and $I = \{p\}$. $I \not\models \overline{\mathbb{L}^{hard}}$ but $P_{\mathbb{P}}(I) = \frac{exp(\alpha)}{exp(\alpha)+exp(\alpha)} = 0.5$.

where

$$\frac{HSAT}{exp(|\mathbb{L}^{hard}| \cdot \alpha)} = \sum_{J \models \overline{\mathbb{L}^{hard}}} exp(\sum_{w:F \in \mathbb{L}_J \setminus \mathbb{L}^{hard}} w)$$

and

$$\begin{aligned}
&\frac{HUNSAT}{exp(|\mathbb{L}^{hard}| \cdot \alpha)} \\
= &\frac{\sum_{J \not\models \overline{\mathbb{L}^{hard}}} exp(|\mathbb{L}^{hard} \cap \mathbb{L}_J| \cdot \alpha) \cdot exp(\sum_{w:F \in \mathbb{L}_J \setminus \mathbb{L}^{hard}} w)}{exp(|\mathbb{L}^{hard}| \cdot \alpha)} \\
= &\sum_{J \not\models \overline{\mathbb{L}^{hard}}} \frac{exp(|\mathbb{L}^{hard} \cap \mathbb{L}_J| \cdot \alpha)}{exp(|\mathbb{L}^{hard}| \cdot \alpha)} \cdot exp(\sum_{w:F \in \mathbb{L}_J \setminus \mathbb{L}^{hard}} w).
\end{aligned}$$

For $J \not\models \overline{\mathbb{L}^{hard}}$, we have $|\mathbb{L}^{hard} \cap \mathbb{L}_J| \leq |\mathbb{L}^{hard}| - 1$, so

$$P_{\mathbb{L}}(I) = \frac{exp(\sum_{w:F \in \mathbb{L}_I \setminus \mathbb{L}^{hard}} w)}{\sum_{J \models \overline{\mathbb{L}^{hard}}} exp(\sum_{w:F \in \mathbb{L}_J \setminus \mathbb{L}^{hard}} w)}.$$

- Suppose I does not satisfy $\overline{\mathbb{L}^{hard}}$. Since $\overline{\mathbb{L}^{hard}}$ is satisfiable, there is at least one interpretation that satisfies $\overline{\mathbb{L}^{hard}}$. Let K denote any such interpretation. We have

$$P_{\mathbb{L}}(I) = \lim_{\alpha \to \infty} \frac{exp(\sum_{w:F \in \mathbb{L}_I} w)}{\sum_{J \in PW} exp(\sum_{w:F \in \mathbb{L}_J} w)}.$$

Splitting the denominator into K and the other interpretations, we have

$$P_{\mathbb{L}}(I) = \lim_{\alpha \to \infty} \frac{exp(\sum_{w:F \in \mathbb{L}_I} w)}{exp(\sum_{w:F \in \mathbb{L}_K} w) + \sum_{J \neq K} exp(\sum_{w:F \in \mathbb{L}_J} w)}.$$

Extracting the weight from formulas in $\mathbb{L}^{hard}$, we have

$$\begin{aligned}
P_{\mathbb{L}}(I) &= \lim_{\alpha \to \infty} \frac{exp(|\mathbb{L}^{hard} \cap \mathbb{L}_I| \cdot \alpha) \cdot exp(\sum_{w:F \in \mathbb{L}_I \setminus \mathbb{L}^{hard}} w)}{exp(|\mathbb{L}^{hard}| \cdot \alpha) \cdot exp(\sum_{w:F \in \mathbb{L}_K \setminus \mathbb{L}^{hard}} w) + \sum_{J \neq K} exp(\sum_{w:F \in \mathbb{L}_J} w)} \\
&\leq \lim_{\alpha \to \infty} \frac{exp(|\mathbb{L}^{hard} \cap \mathbb{L}_I| \cdot \alpha) \cdot exp(\sum_{w:F \in \mathbb{L}_I \setminus \mathbb{L}^{hard}} w)}{exp(|\mathbb{L}^{hard}| \cdot \alpha) \cdot exp(\sum_{w:F \in \mathbb{L}_K \setminus \mathbb{L}^{hard}} w)}.
\end{aligned}$$

Since I does not satisfy $\overline{\mathbb{L}^{hard}}$, $|\mathbb{L}^{hard} \cap \mathbb{L}_I| \leq |\mathbb{L}^{hard}| - 1$, and thus

$$P_{\mathbb{L}}(I) \leq \lim_{\alpha\to\infty} \frac{exp(|\mathbb{L}^{hard} \cap \mathbb{L}_I| \cdot \alpha) \cdot exp(\sum_{w:F\in \mathbb{L}_I \setminus \mathbb{L}^{hard}} w)}{exp(|\mathbb{L}^{hard}| \cdot \alpha) \cdot exp(\sum_{w:F\in \mathbb{L}_K \setminus \mathbb{L}^{hard}} w)} = 0.$$

□

For any LP$^{\text{MLN}}$ program Π, define MLN program $\mathbb{L}_\Pi$ to be the union of Π and $\{\alpha : LF_{\overline{\Pi}}(L) \mid L \text{ is a loop of } \overline{\Pi}\}$.

Lemma 7. *For any* LP$^{\text{MLN}}$ *program* Π *and any interpretation* I*, if* $I \in SM'[\Pi]$*, then* $I \models LF_{\overline{\Pi}}$.

Proof. Suppose $I \in SM'[\Pi]$, then $I \models_{\text{SM}} \overline{\Pi^{hard}} \cup \overline{(\Pi^{soft})_I}$, which implies $I \models_{\text{SM}} \overline{\Pi_I}$, and further implies $I \models LF_{\overline{\Pi_I}}$. By Lemma 5, $I \models LF_{\overline{\Pi}}$. □

Theorem 8 *For any* LP$^{\text{MLN}}$ *program* Π *such that* $SM'[\Pi]$ *is not empty,* Π *and* $\mathbb{L}_\Pi$ *have the same probability distribution over all interpretations, and consequently, the stable models of* Π *and the models of* $\mathbb{L}_\Pi$ *coincide.*

Proof. We will show that $P_\Pi(I) = P_{\mathbb{L}_\Pi}(I)$ for all interpretations I. Since $SM'[\Pi]$ is not empty, by Lemma 7, there exists at least one interpretation J such that $J \models LF_{\overline{\Pi}}$.

- Suppose I is a stable model of $\overline{\Pi_I}$. By definition,

$$P_{\mathbb{L}_\Pi}(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_I}} w_i)}{\sum_{J\in PW} exp(\sum_{r_i\in \overline{(\mathbb{L}_\Pi)_J}} w_i)}.$$

Splitting the denominator into interpretations that satisfy $LF_{\overline{\Pi}}$ and those that do not, we get

$$P_{\mathbb{L}_\Pi}(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_I}} w_i)}{\sum_{J\models LF_{\overline{\Pi}}} exp(\sum_{r_i\in \overline{(\mathbb{L}_\Pi)_J}} w_i) + \sum_{J\not\models LF_{\overline{\Pi}}} exp(\sum_{r_i\in \overline{(\mathbb{L}_\Pi)_J}} w_i)}.$$

Extracting the weights from the formulas in $LF_{\overline{\Pi}}$, we get

$$P_{\mathbb{L}_\Pi}(I) = \lim_{\alpha\to\infty} \frac{exp(|LF_{\overline{\Pi}}| \cdot \alpha) \cdot exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_I} \setminus LF_{\overline{\Pi}}} w_i)}{\sum_{J \models LF_{\overline{\Pi}}} exp(|LF_{\overline{\Pi}}| \cdot \alpha) \cdot exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_J} \setminus LF_{\overline{\Pi}}} w_i) + \sum_{J \not\models LF_{\overline{\Pi}}} exp(|\overline{(\mathbb{L}_\Pi)_J} \cap LF_{\overline{\Pi}}| \cdot \alpha) \cdot exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_J} \setminus LF_{\overline{\Pi}}} w_i)}.$$

Dividing both the numerator and the denominator by $exp(|LF_\Pi| \cdot \alpha)$, we have

$$P_{\mathbb{L}_\Pi}(I) = \lim_{\alpha\to\infty} \frac{exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_I} \setminus LF_{\overline{\Pi}}} w_i)}{\sum_{J \models LF_{\overline{\Pi}}} exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_J} \setminus LF_{\overline{\Pi}}} w_i) + \sum_{J \not\models LF_{\overline{\Pi}}} \frac{exp(|\overline{(\mathbb{L}_\Pi)_J} \cap LF_{\overline{\Pi}}| \cdot \alpha)}{exp(|LF_{\overline{\Pi}}| \cdot \alpha)} \cdot exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_J} \setminus LF_{\overline{\Pi}}} w_i)}.$$

For those J that do not satisfy $LF_{\overline{\Pi}}$, $|\overline{(\mathbb{L}_\Pi)_J} \cap LF_{\overline{\Pi}}| \leq |LF_{\overline{\Pi}}| - 1$. So

$$\lim_{\alpha\to\infty} \frac{exp(|\overline{(\mathbb{L}_\Pi)_J} \cap LF_{\overline{\Pi}}| \cdot \alpha)}{exp(|LF_{\overline{\Pi}}| \cdot \alpha)} = 0$$

. Consequently

$$P_{\mathbb{L}_\Pi}(I) = \frac{exp(\sum_{r_i \in \overline{(\mathbb{L}_\Pi)_I} \setminus LF_{\overline{\Pi}}} w_i)}{\sum_{J \models LF_{\overline{\Pi}}} exp(\sum_{r_i \in \overline{(\mathbb{L}_{Pi})_J} \setminus LF_{\overline{\Pi}}} w_i)}.$$

From the construction of $\mathbb{L}_\Pi$ it can be easily seen that $\overline{(\mathbb{L}_\Pi)_K} \setminus LF_{\overline{\Pi}} = \overline{\Pi_K}$ for all interpretations K. So

$$P_{\mathbb{L}_\Pi}(I) = \frac{exp(\sum_{r_i \in \overline{\Pi_I}} w_i)}{\sum_{J \models LF_{\overline{\Pi}}} exp(\sum_{r_i \in \overline{\Pi_J}} w_i)}.$$

By Lemma 5, for any $J \models LF_{\overline{\Pi}}$, we have $J \models LF_{\overline{\Pi_J}}$ and thus J is a stable model of $\overline{\Pi_J}$. So

$$\begin{aligned} P_{\mathbb{L}_\Pi}(I) &= \frac{exp(\sum_{r_i \in \overline{\Pi_I}} w_i)}{\sum_{J \models_{SM} \overline{\Pi_J}} exp(\sum_{r_i \in \overline{\Pi_J}} w_i)} \\ &= \frac{W_\Pi(I)}{\sum_{J \in SM[\Pi]} W_\Pi(J)} \\ &= P_\Pi(I). \end{aligned}$$

- Suppose I is not a stable model of $\overline{\Pi_I}$. Then $P_\Pi(I) = 0$. On the other hand, since $I \models \overline{\Pi_I}$ by definition, it must be the case that $I \not\models LF_{\overline{\Pi_I}}$. By Lemma 5, $I \not\models LF_{\overline{\Pi}}$. So there is at least one subset L of σ such that $I \not\models LF_{\overline{\Pi}}(L)$. Clearly $\alpha : LF_{\overline{\Pi}}(L) \in \mathbb{L}_\Pi$ and $LF_{\overline{\Pi}}(L) \in \overline{(\mathbb{L}_\Pi)^{hard}}$. So $I \not\models \overline{(\mathbb{L}_\Pi)^{hard}}$. From the construction of $\mathbb{L}_\Pi$ we can see that $\overline{(\mathbb{L}_\Pi)^{hard}} = \overline{\Pi^{hard}} \cup LF_{\overline{\Pi}}$. Since $SM'[\Pi]$ is not empty, there is at least one interpretation J such that $J \models_{\text{SM}} \overline{\Pi^{hard}} \cup \overline{(\Pi^{soft})_J}$. This interpretation J satisfies $LF_{\overline{\Pi^{hard} \cup (\Pi^{soft})_J}}$. By Lemma 7, J satisfies $LF_{\overline{\Pi}}$. So J satisfies $\overline{\Pi^{hard}} \cup LF_{\overline{\Pi}}$ and thus $(\mathbb{L}_\Pi)^{hard}$ is satisfiable. By Lemma 6, $P_{\mathbb{L}_\Pi}(I) = 0$.

$\square$

4.7.5 Proof of Theorem 10

In this section and the next section, we write $\sum_x f(x)$, where f is some function over a Boolean variable, as a shorthand of

$$\sum_{x \in \{true, false\}} f(x),$$

and write

$$\sum_{x_1, \ldots, x_m} f(x_1, \ldots, x_m)$$

as a shorthand of

$$\sum_{x_1} \sum_{x_2} \cdots \sum_{x_m} f(x_1, \ldots, x_m).$$

Given a ProbLog program $\mathbb{P}$, let $PA_{\mathbb{P}}$ denote the set of all probabilistic atoms in $\mathbb{P}$. We say a subset TC of $PA_{\mathbb{P}}$ is the total choice of an interpretation I if for all $p \in TC$, $I \models p$ and for all $q \in PA_{\mathbb{P}} \setminus TC$, $I \not\models q$.

Lemma 8. *For any ProbLog program* $\mathbb{P}$*,*

$$\sum_{TC \subseteq PA_{\mathbb{P}}} Pr_{\mathbb{P}}(TC) = 1.$$

Proof. Suppose $PA_{\mathbb{P}} = \{a_1, a_2, \ldots, a_k\}$.

$$\begin{aligned} &\sum_{TC \subseteq PA_{\mathbb{P}}} Pr_{\mathbb{P}}(TC) \\ = &\sum_{TC \subseteq PA_{\mathbb{P}}} \Big(\prod_{a_i \in TC} p_i \cdot \prod_{a_j \in PA_{\mathbb{P}} \setminus TC} (1 - p_j) \Big). \end{aligned}$$

Let $\mathbf{p_i}(x)$ where $x \in \{\mathbf{t}, \mathbf{f}\}$ be defined as

$$\mathbf{p_i}(x) = \begin{cases} p_i & if\ x = \mathbf{t} \\ 1 - p_i & if\ x = \mathbf{f} \end{cases}.$$

Clearly $\sum_a \mathbf{p_i}(a) = 1$ for any $i \in \{1, \dots, k\}$. $\sum_{TC \subseteq PA_{\mathbb{P}}} Pr_{\mathbb{P}}(TC)$ can be rewritten as

$$\begin{aligned} &\sum_{TC \subseteq PA_{\mathbb{P}}} Pr_{\mathbb{P}}(TC) \\ =&\sum_{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k} \mathbf{p_i}(\mathbf{a}_i) \cdot \dots \cdot \mathbf{p_i}(\mathbf{a}_i) \end{aligned}$$

where $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$ are Boolean variables representing whether or not $a_i \in TC$, i.e., $\mathbf{a_i} = \mathbf{t}$ if $a_i \in TC$, $\mathbf{a_i} = \mathbf{f}$ otherwise. Rearranging the equation we have

$$\begin{aligned} &\sum_{C \subseteq PA_{\mathbb{P}}} Pr_{\mathbb{P}}[C] \\ =&\sum_{\mathbf{a}_1} \mathbf{p_1}(\mathbf{a}_1) \sum_{\mathbf{a}_2} \mathbf{p_2}(\mathbf{a}_2) \cdot \sum_{\mathbf{a}_k} \mathbf{p_k}(\mathbf{a}_k) \\ =&1. \end{aligned}$$

□

Theorem 13. *When a (deterministic) program* Π *has a total well-founded model, then this model is also the single stable model of* Π.

Proof. Proven in Van Gelder *et al.* (1991). □

Lemma 9. *Let* $\mathbb{P} = \langle PF, \Pi \rangle$ *be any ProbLog program that does not contain any probabilistic atom for which the probability is* 0 *or* 1. $\mathbb{P}$ *and its* LP$^{\text{MLN}}$ *representation* $\Pi_{\mathbb{P}}$ *have the same probability distribution over all interpretations.*

Proof. Since $\mathbb{P}$ is a well-defined ProbLog program, for all $TC \subseteq PA_{\mathbb{P}}$, $TC \cup \Pi$ has one total well-founded model. Let $TC(I)$ denote the total choice of I.

- Suppose I is the total well-founded model of $TC(I) \cup \Pi$. According to the definition,

$$P_{\mathbb{P}}(I) = Pr_{\mathbb{P}}(TC(I))$$
$$= \prod_{a_i \in TC(I)} p_i \cdot \prod_{b_j \in PA_{\mathbb{P}} \setminus TC(I)} (1 - p_j).$$

By Theorem 13, I is also the unique stable model of $TC(I) \cup \Pi$. It can be seen that I is the only stable model of $TC(I) \cup \Pi \cup \{\leftarrow p \mid p \notin TC(I)\}$, which is $\overline{(\Pi_{\mathbb{P}})_I}$. Clearly $\overline{(\Pi_{\mathbb{P}})^{hard}} = \Pi \subseteq \overline{(\Pi_{\mathbb{P}})_I}$ and consequently $I \models_{\text{SM}} \overline{(\Pi_{\mathbb{P}})^{hard}} \cup \overline{(\Pi_{\mathbb{P}})_I^{soft}}$. By Proposition 3,

$$P_{\Pi_{\mathbb{P}}}(I) = \frac{exp(\sum_{F_i \in \overline{(\Pi_{\mathbb{P}})_I} \setminus \overline{(\Pi_{\mathbb{P}})^{hard}}} w_i)}{\sum_{J \in SM'[\Pi_{\mathbb{P}}]} exp(\sum_{F_i \in \overline{(\Pi_{\mathbb{P}})_J} \setminus \overline{(\Pi_{\mathbb{P}})^{hard}}} w_i)}$$
$$= \frac{exp(\sum_{a_i \in PA_{\mathbb{P}} : I \models a_i} ln(p_i) + \sum_{a_i \in PA_{\mathbb{P}} : I \not\models a_i} ln(1 - p_i))}{\sum_{J \in SM'[\Pi_{\mathbb{P}}]} exp(\sum_{a_i \in PA_{\mathbb{P}} : J \models a_i} ln(p_i) + \sum_{a_i \in PA_{\mathbb{P}} : J \not\models a_i} ln(1 - p_i))}$$
$$= \frac{\prod_{a_i \in TC(I)} p_i \prod_{a_i \notin TC(I)} (1 - p_i)}{\sum_{J \in SM'[\Pi_{\mathbb{P}}]} \prod_{a_i \in TC(J)} p_i \prod_{a_i \notin TC(J)} (1 - p_i)}.$$

Clearly for every J such that $J \in SM'[\Pi_{\mathbb{P}}]$, there is a total choice $TC(J)$. And since the ProbLog program $\mathbb{P}$ is well-defined, for every total choice TC' there is a total well-founded model of $TC' \cup \Pi$. By Theorem 13, this means for every total choice C there is a unique stable model of $C \cup \Pi$. It can be seen that this stable model is also the unique stable model of $TC' \cup \Pi \cup \{\neg p \mid p \notin TC(I)\}$. So

$$P_{\Pi_{\mathbb{P}}}(I) = \frac{\prod_{a_i \in TC(I)} p_i \prod_{a_i \notin TC(I)} (1 - p_i)}{\sum_{TC' \subseteq PA_{\mathbb{P}}} \prod_{a_i \in TC'} p_i \prod_{a_i \notin TC'} (1 - p_i)}$$
$$= \frac{\prod_{a_i \in TC(I)} p_i \prod_{a_i \notin TC(I)} (1 - p_i)}{\sum_{TC' \subseteq PA_{\mathbb{P}}} Pr_{\mathbb{P}}(TC')}.$$

By Lemma 8, the denominator equals 1, so

$$
\begin{aligned}
P_{\Pi_{\mathbb{P}}}(I) &= \prod_{a_i \in TC(I)} p_i \prod_{a_i \notin TC(I)} (1 - p_i) \\
&= P_{\mathbb{P}}(I).
\end{aligned}
$$

- Suppose I is not the total well-founded model of $TC(I) \cup \Pi$. Then $P_{\mathbb{P}}(I) = 0$. Since $\mathbb{P}$ is well-defined. The total well-founded model J of $TC(I) \cup \Pi$ exists and by Theorem 13, J is also the unique stable model of $TC(I) \cup \Pi$. It must be the case that $I \neq J$ and thus I cannot be a stable model of $TC(I) \cup \Pi$. There are following two cases:

  – Suppose $I \not\models TC(I) \cup \Pi$. Since $TC(I)$ is the total choice of I, $I \models TC(I)$. It follows that $I \not\models \Pi$, i.e., there is at least one rule $F \in \Pi$ such that $I \not\models F$. According to the definition, $\alpha : F \in (\Pi_{\mathbb{P}})^{hard}$. By Proposition 3, $P_{\Pi_{\mathbb{P}}}(I) = 0$.

  – Suppose $I \models TC(I) \cup \Pi$ but I is not a stable model of $TC(I) \cup \Pi$. By Theorem 2, it follows that there must be at least one loop L of $\overline{\Pi}$ such that $I \models L^{\wedge}$ but $I \not\models ES_{TC(I) \cup \Pi}(L)$. It can be seen that

  $$(\Pi_{\mathbb{P}})_I = TC(I) \cup \Pi \cup \{\leftarrow p \mid p \notin TC(I)\}.$$

  It can be seen that $ES_{(\Pi_{\mathbb{P}})_I}(L) = ES_{TC(I) \cup \Pi}(L)$. It follows that $I \not\models_{SM} \overline{\mathbb{P}'_I}$. So $W_{\Pi_{\mathbb{P}}}(I) = 0$ and thus $P_{\Pi_{\mathbb{P}}}(I) = 0$.

□

Theorem 10 *Any (well-defined) ProbLog program $\mathbb{P}$ and its LP^{MLN} representation $\Pi_{\mathbb{P}}$ have the same probability distribution over all interpretations.*

Proof. We first convert $\mathbb{P} = \langle PF, \Pi \rangle$ into a ProbLog program that does not contain any probabilistic atom for which the probability is 0 or 1 as follows.

- For each probabilistic atom p such that $pr(p) = 0$:
  - Remove all the rules in Π where p occurs in the body positively (i.e., as the literal p);
  - Remove all the literals *not* p that occurs in Π.
- For each probabilistic atom q such that $pr(q) = 1$:
  - Remove all the literals p that occurs in Π;
  - Remove all the rules in Π where p occurs in the body negatively (i.e., as the literal *not* p).

Let $T(\mathbb{P})$ denote the program obtained from $\mathbb{P}$ as above. Clearly $T(\mathbb{P})$ specifies the same probability distribution as $\mathbb{P}$, if we restrict attention to atoms other than those atoms for which the probability is 0 or 1. By Lemma 9, $T(\mathbb{P})$ and its LP$^{\text{MLN}}$ representation $\Pi_{T(\mathbb{P})}$ have the same probability distribution over all interpretations. From the construction of $T(\mathbb{P})$, it can be seen that $\Pi_{\mathbb{P}}$ specifies the same probability distribution as $\Pi_{T(\mathbb{P})}$ if we restrict attention to atoms other than those atoms for which the probability is 0 or 1. Also it is clearly that those atoms in $\mathbb{P}$ for which the probability is 0 or 1 have exactly the same constant truth values as these atoms in $T(\mathbb{P})$. So $\mathbb{P}$ and its LP$^{\text{MLN}}$ representation $\Pi_{\mathbb{P}}$ have the same probability distribution over all interpretations. $\square$

4.7.6 Proof of Theorem 11

Lemma 10. *Let $\mathbb{L}$ be an MLN, and let $\mathbb{L}_\alpha = \{F \mid \alpha : F \in \mathbb{L}\}$. When $\mathbb{L}_\alpha$ is satisfiable,*

- *if I satisfies $\mathbb{L}_\alpha$,*

$$Pr_{\mathbb{L}}[I] = \frac{exp(\sum_{(w:F)\in\mathbb{L}\ :\ F\ \in\ \mathbb{L}_I\setminus\mathbb{L}_\alpha} w)}{\sum_{J\in PW: J\models\mathbb{L}_\alpha} exp(\sum_{(w:F)\in\mathbb{L}\ :\ F\ \in\ \mathbb{L}_J\setminus\mathbb{L}_\alpha} w)}$$

- *otherwise, $Pr_{\mathbb{L}}[I] = 0$.*[10]

Proof. For any interpretation I, by definition, we have

$$\begin{aligned} Pr_{\mathbb{L}}\,[I] &= \lim_{\alpha\to\infty} \frac{W_{\mathbb{L}}(I)}{\sum_{J\in PW} W_{\mathbb{L}}(J)} \\ &= \lim_{\alpha\to\infty} \frac{W_{\mathbb{L}}(I)}{\sum_{J\in PW} exp(\sum_{F_i\in\mathbb{L}_J} w_i)} \end{aligned}$$

- Suppose I satisfies $\mathbb{L}_\alpha$. We have

$$Pr_{\mathbb{L}}\,[I] = \lim_{\alpha\to\infty} \frac{exp(\sum_{F_i\in\mathbb{L}_I} w_i)}{\sum_{J\in PW} exp(\sum_{F_i\in\mathbb{L}_J} w_i)}.$$

Splitting the denominator into two parts: those J that satisfies $\mathbb{L}_\alpha$ and those that do not, and extracting the weight of formulas in $\mathbb{L}_\alpha$, we have

$$Pr_{\mathbb{L}}\,[I] = \lim_{\alpha\to\infty} \frac{exp(|\mathbb{L}_\alpha|\cdot\alpha)\cdot exp(\sum_{F_i\in\mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{exp(|\mathbb{L}_\alpha|\cdot\alpha)\cdot\sum_{J\models\mathbb{L}_\alpha} exp(\sum_{F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i) + \sum_{J\not\models\mathbb{L}_\alpha} exp(|\mathbb{L}_\alpha\cap\mathbb{L}_J|\cdot\alpha)\cdot exp(\sum_{F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i)}.$$

We divide both the numerator and the denominator by $exp(|\mathbb{L}_\alpha|\cdot\alpha)$.

$$\begin{aligned} Pr_{\mathbb{L}}\,[I] &= \lim_{\alpha\to\infty} \frac{exp(\sum_{F_i\in\mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{\sum_{J\models\mathbb{L}_\alpha} exp(\sum_{F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i) + \frac{\sum_{J\not\models\mathbb{L}_\alpha} exp(|\mathbb{L}_\alpha\cap\mathbb{L}_J|\cdot\alpha)\cdot exp(\sum_{F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i)}{exp(|\mathbb{L}_\alpha|\cdot\alpha)}} \\ &= \lim_{\alpha\to\infty} \frac{exp(\sum_{F_i\in\mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{\sum_{J\models\mathbb{L}_\alpha} exp(\sum_{F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i) + \sum_{J\not\models\mathbb{L}_\alpha} \frac{exp(|\mathbb{L}_\alpha\cap\mathbb{L}_J|\cdot\alpha)}{exp(|\mathbb{L}_\alpha|\cdot\alpha)}\cdot exp(\sum_{F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i)}. \end{aligned}$$

For $J\not\models\mathbb{L}_\alpha$, $|\mathbb{L}_\alpha\cap\mathbb{L}_J| \leq |\mathbb{L}_\alpha| - 1$, so

$$Pr_{\mathbb{L}}\,[I] = \frac{exp(\sum_{w_i:F_i\in\mathbb{L}:F_i\in\mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{\sum_{J\models\mathbb{L}_\alpha} exp(\sum_{w_i:F_i\in\mathbb{L}:F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i)}.$$

[10]This proposition does not hold when $\mathbb{L}_\alpha$ is not satisfiable. For example, consider $\mathbb{L} = \{\alpha : p, \alpha :\leftarrow p\}$ and $I = \{p\}$. $I\not\models\mathbb{L}_\alpha$ but $Pr_{\mathbb{P}}\,[I] = \frac{exp(\alpha)}{exp(\alpha)+exp(\alpha)} = 0.5$.

- Suppose I does not satisfy $\mathbb{L}_\alpha$. Since $\mathbb{L}_\alpha$ is satisfiable, there is at least one interpretation that satisfies $\mathbb{L}_\alpha$. Let K denote any such interpretation. We have

$$Pr_{\mathbb{L}}[I] = \lim_{\alpha\to\infty} \frac{exp(\sum_{F_i \in \mathbb{L}_I} w_i)}{\sum_{J\in PW} exp(\sum_{F_i\in\mathbb{L}_J} w_i)}.$$

Splitting the denominator into K and the other interpretations, we have

$$Pr_{\mathbb{L}}[I] = \lim_{\alpha\to\infty} \frac{exp(\sum_{F_i \in \mathbb{L}_I} w_i)}{exp(\sum_{F_i\in\mathbb{L}_K} w_i) + \sum_{J\neq K} exp(\sum_{F_i\in\mathbb{L}_J} w_i)}.$$

Extracting the weight from formulas in $\mathbb{L}_\alpha$, we have

$$\begin{aligned} Pr_{\mathbb{L}}[I] &= \lim_{\alpha\to\infty} \frac{exp(|\mathbb{L}_\alpha \cap \mathbb{L}_I|\cdot\alpha)\cdot exp(\sum_{F_i \in \mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{exp(|\mathbb{L}_\alpha|\cdot\alpha)\cdot exp(\sum_{F_i\in\mathbb{L}_K} w_i) + \sum_{J\neq K} exp(\sum_{F_i\in\mathbb{L}_J} w_i)} \\ &\leq \lim_{\alpha\to\infty} \frac{exp(|\mathbb{L}_\alpha \cap \mathbb{L}_I|\cdot\alpha)\cdot exp(\sum_{F_i \in \mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{exp(|\mathbb{L}_\alpha|\cdot\alpha)\cdot exp(\sum_{F_i\in\mathbb{L}_K} w_i)}. \end{aligned}$$

Since I does not satisfy $\mathbb{L}_\alpha$, $|\mathbb{L}_\alpha \cap \mathbb{L}_I| \leq |\mathbb{L}_\alpha| - 1$, and thus

$$Pr_{\mathbb{L}}[I] \leq \lim_{\alpha\to\infty} \frac{exp(|\mathbb{L}_\alpha \cap \mathbb{L}_I|\cdot\alpha)\cdot exp(\sum_{F_i \in \mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{exp(|\mathbb{L}_\alpha|\cdot\alpha)\cdot exp(\sum_{F_i\in\mathbb{L}_K} w_i)} = 0.$$

□

Lemma 11. *The solution of a probabilistic causal model* $\mathbb{M} = \langle\langle U, V, F\rangle, P(U)\rangle$ *are identical to the models of* $Comp(\Pi_{\mathbb{M}})$ *and their probability distributions coincide.*

Proof. First, we notice from the construction of $\Pi_{\mathbb{M}}$ from $\mathbb{M}$ and the construction of $Comp(\Pi_{\mathbb{M}})$ from $\Pi_{\mathbb{M}}$ that

$$\begin{aligned} Comp(\Pi_{\mathbb{M}}) = \;& \{ln(P(U_i = \mathbf{t})) :\; U_i, ln(P(U_i = \mathbf{f})) : \leftarrow U_i \mid U_i \in U\} \cup \\ & \{\alpha \;:\; V_i \leftarrow F_i \mid V_i = F_i \in F\} \cup \\ & \{\alpha \;:\; V_i \rightarrow F_i \mid V_i = F_i \in F\} \cup \\ & \{\alpha \;:\; U_i \rightarrow \top \mid U_i \in U\}. \end{aligned}$$

- (From a solution of $\mathbb{M}$ to a model of $Comp(\Pi_{\mathbb{M}})$) Consider an interpretation I of $U \cup V$ which is a solution of $\mathbb{M}$. By definition, I satisfies $V_i \leftrightarrow F_i$ for all equations $V_i = F_i$ in F. It follows that I satisfies $Comp(\Pi_{\mathbb{M}})_\alpha$. By Lemma 10, I receives nonzero probability, which means I is a model of $Comp(\Pi_{\mathbb{M}})$. Furthermore, the probability of I under $Comp(\Pi_{\mathbb{M}})$ is

$$\begin{aligned} Pr_{Comp(\Pi_{\mathbb{M}})}[I] &= \frac{exp(\sum_{(w:F)\in Comp(\Pi_{\mathbb{M}}):F\in Comp(\Pi_{\mathbb{M}})_I \setminus Comp(\Pi_{\mathbb{M}})_\alpha} w)}{\sum_{J\in PW:J\models Comp(\Pi_{\mathbb{M}})_\alpha} exp(\sum_{(w:F)\in Comp(\Pi_{\mathbb{M}}):F\in Comp(\Pi_{\mathbb{M}})_J \setminus Comp(\Pi_{\mathbb{M}})_\alpha} w)} \\ &= \frac{exp(\sum_{I\models U_i} ln(P(U_i{=}\mathbf{t})) + \sum_{I\not\models U_i} ln(P(U_i{=}\mathbf{f})))}{\sum_{J\in PW:J\models Comp(\Pi_{\mathbb{M}})_\alpha} exp(\sum_{(w:F)\in Comp(\Pi_{\mathbb{M}}):F\in Comp(\Pi_{\mathbb{M}})_J \setminus Comp(\Pi_{\mathbb{M}})_\alpha} w)} \end{aligned}$$

  Since $\mathbb{M}$ is a probabilistic causal model, the valuation of endogenous atoms are uniquely determined by the valuation of exogenous atoms, and for every valuation of exogenous atoms there exists a valuation of endogenous atoms. Also it can be seen that there are $2^{|U|}$ interpretations of $Comp(\Pi_{\mathbb{M}})$ that satisfy $Comp(\Pi_{\mathbb{M}})_\alpha$, corresponding to all possible valuations of atoms in U. So the denominator can be further rewritten as

$$\begin{aligned} Pr_{Comp(\Pi_{\mathbb{M}})}[I] &= \frac{exp(\sum_{I\models U_i} ln(P(U_i{=}\mathbf{t})) + \sum_{I\not\models U_i} ln(P(U_i{=}\mathbf{f})))}{\sum_{J\in PW:J\models Comp(\Pi_{\mathbb{M}})_\alpha} exp(\sum_{J\models U_i} ln(P(U_i{=}\mathbf{t})) + \sum_{J\not\models U_i} ln(P(U_i{=}\mathbf{f})))} \\ &= \frac{\prod_{I\models U_i} P(U_i{=}\mathbf{t}) \cdot \prod_{I\not\models U_i} P(U_i{=}\mathbf{f})}{\sum_{J\in PW:J\models Comp(\Pi_{\mathbb{M}})_\alpha} \prod_{J\models U_i} P(U_i{=}\mathbf{t}) \cdot \prod_{J\not\models U_i} P(U_i{=}\mathbf{f})} \\ &= \frac{\prod_{I\models U_i} P(U_i{=}\mathbf{t}) \cdot \prod_{I\not\models U_i} P(U_i{=}\mathbf{f})}{1} \\ &= P(U = I_U) \\ &= P_{\mathbb{M}}(I). \end{aligned}$$

- (From a model of $Comp(\Pi_{\mathbb{M}})$ to a solution of $\mathbb{M}$) Consider a model I of $Comp(\Pi_{\mathbb{M}})$. By Proposition 4 (in the technical appendix), I satisfies $Comp(\Pi_{\mathbb{M}})_\alpha$, which means I satisfies all the equivalence $V_i \leftrightarrow F_i$ for all equation $V_i = F_i$ in F. Since $\mathbb{M}$ is a probabilistic causal model, it must be the case that no other interpretation J such that $J \neq^V I$ satisfies all such equivalences. So I is a solution of $\mathbb{M}$. As shown in the first bullet, the probability of I under $Comp(\Pi_{\mathbb{M}})$ is

$$Pr_{Comp(\Pi_{\mathbb{M}})}[I] = P(U = I_U)$$

which coincides with the probability of I under $\mathbb{M}$.

$\square$

Proposition 7. *The solutions of a probabilistic causal model* $\mathbb{M}$ *are identical to the stable models of* $\Pi_{\mathbb{M}}$ *and their probability distributions coincide.*

Proof. Since $\mathbb{M}$ is a probabilistic causal model, for $\Pi_{\mathbb{M}}$ the following conditions hold

- $Comp(\Pi_{\mathbb{M}})_\alpha$ is satisfiable, and
- Π is tight.

By Theorem 9, the stable model of $\Pi_{\mathbb{M}}$ are identical to the models of $Comp(\Pi_{\mathbb{M}})$ and their probability distributions coincide; by Lemma 11, the solutions of M are identical to the models of $Comp(\Pi_{\mathbb{M}})$ and their probability distributions coincide. In conclusion, the solutions of M are identical to the stable models of $\Pi_{\mathbb{M}}$ and their probability distributions coincide. $\square$

We say two MLN programs are *equivalent* to each other if they have the same probability distribution over possible worlds.

Lemma 12. *For any MLN program* $\mathbb{L}$*, when* $\mathbb{L}_\alpha$ *is satisfiable,* $\mathbb{L}_\alpha$ *can be replaced by any set* $\mathbb{L}'_\alpha$ *of hard formulas that is classically equivalent to* $\mathbb{L}_\alpha$ *without changing the models and probability distribution.*[11]

Proof. Let $\mathbb{L}'$ denote the MLN program obtained by replacing $\mathbb{L}_\alpha$ with $\mathbb{L}'_\alpha$. Consider any interpretation I of the underlying signature.

[11] This lemma does not hold when $\mathbb{L}_\alpha$ is not satisfiable. For example, $\mathbb{L}_1 = \{\alpha \ : \ p, \ \alpha \ : \ \neg p\}$ and $\mathbb{L}_2 = \{\alpha \ : \ p \wedge \neg p\}$ has different probability distributions over possible worlds.

- Suppose I satisfies $\mathbb{L}_\alpha$. By Lemma 10, we have

$$Pr_{\mathbb{L}}[I] = \frac{exp(\sum_{(w_i:F_i)\in\mathbb{L}:F_i\in\mathbb{L}_I\setminus\mathbb{L}_\alpha} w_i)}{\sum_{J\in PW:J\models\mathbb{L}_\alpha} exp(\sum_{(w_i:F_i)\in\mathbb{L}:F_i\in\mathbb{L}_J\setminus\mathbb{L}_\alpha} w_i)}$$

  Since $\mathbb{L}\setminus\mathbb{L}_\alpha = \mathbb{L}'\setminus\mathbb{L}'_\alpha$ and $\mathbb{L}_\alpha$ is equivalent to $\mathbb{L}'_\alpha$, we have

$$\begin{aligned} Pr_{\mathbb{L}}[I] &= \frac{exp(\sum_{(w_i:F_i)\in\mathbb{L}':F_i\in\mathbb{L}'_I\setminus\mathbb{L}'_\alpha} w_i)}{\sum_{J\in PW:J\models\mathbb{L}'_\alpha} exp(\sum_{(w_i:F_i)\in\mathbb{L}':F_i\in\mathbb{L}'_J\setminus\mathbb{L}'_\alpha} w_i)} \\ &= Pr_{\mathbb{L}'}[I]. \end{aligned}$$

- Suppose I does not satisfy $\mathbb{L}_\alpha$. Since $\mathbb{L}_\alpha$ is equivalent to $\mathbb{L}'_\alpha$, I does not satisfy $\mathbb{L}'_\alpha$. By Proposition 4 (in the technical appendix), we have $Pr_{\mathbb{L}}[I] = Pr_{\mathbb{L}'}[I] = 0$.

□

Theorem 11 *Given any $Y \subseteq V$ and variable assignments $X = x$, $Y = y$, $Z = z$, the probability defined by PCM, $P_{\mathbb{M}}(Y_{X=x} = y \mid Z = z)$, is equal to the following probability defined by* LP$^{\text{MLN}}$ *semantics,*

$$\begin{aligned} P_{\mathbb{M}}(Y_{X=x} = y \mid Z = z) &= P_{\Pi_{\mathbb{M}}}(Do(X = x, counterfactual) \wedge Y(counterfacutual) = y \mid Z(actual) = z) \\ &= \frac{\sum_{I\models Do(X=x,counterfactual)\wedge Y(counterfacutual)=y\wedge Z(actual)=z} P(I)}{\sum_{I\models Z(actual)=z} P(I)} \end{aligned}$$

where $Do(X = x, counterfactual)$ is an abbreviation of

$$Do(x_{1X_1}, counterfactual) \wedge \cdots \wedge Do(x_{nX_n}, counterfactual)$$

for $X = \langle X_1, \ldots, X_n\rangle$ and $x = \langle x_1, \ldots, x_n\rangle$, and similarly $V(w) = v$ where V is Y or Z, v is y or z and w is $actual$ or $counterfactual$ is an abbreviation of

$$V_1(w) = v_1 \wedge \cdots \wedge V_n(w) = v_n$$

for $V = \langle V_1, \ldots, V_n\rangle$ and $v = \langle v_1, \ldots, v_n\rangle$.

Proof. By the assumption that submodels are causal models, and from the construction of $\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x)$, it can be seen that for $\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x)$ the following conditions hold

- $Comp(\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x))_\alpha$ is satisfiable, and
- $\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x)$ is tight.

Thus by Theorem 9 we have that that stable models of $\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x)$ are identical to the models of $Comp(\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x))$ and their probability distributions coincide. It can be seen that

$$
\begin{aligned}
&Comp(\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x)) = \\
&\{ln(P(U_i = \mathbf{t})) :\ U_i, ln(P(U_i = \mathbf{f})) : \leftarrow U_i \mid U_i \in U\} \cup \\
&\{\alpha\ :\ U_i \rightarrow \top \mid U_i \in U\} \cup \\
&\{\alpha\ :\ V_i \leftarrow F_i \mid V_i \in V\} \cup \\
&\{\alpha\ :\ V_i \rightarrow F_i \mid V_i \in V\} \cup \\
&\{\alpha\ :\ V_i^* \leftarrow F_i^* \wedge \neg\, Do(V_i = \mathbf{t}) \wedge \neg\, Do(V_i = \mathbf{f}) \mid V_i \in V\} \cup \\
&\{\alpha\ :\ V_i^* \leftarrow Do(V_i = \mathbf{t}) \mid V_i \in V\} \cup \\
&\{\alpha\ :\ Do(X_i = x_i) \mid X_i \in X, x_i \in x\} \cup \\
&\{\alpha\ :\ V_i^* \rightarrow (F_i^* \wedge \neg\, Do(V_i = \mathbf{t}) \wedge \neg\, Do(V_i = \mathbf{f})) \vee Do(V_i = \mathbf{t}) \mid V_i \in V\} \\
&\{\alpha\ :\ V_i^* \leftarrow F_i^* \wedge \neg\, Do(V_i = \mathbf{t}) \wedge \neg\, Do(V_i = \mathbf{f}) \mid V_i \in V\} \cup \\
&\{\alpha\ :\ Do(X_i = x_i) \rightarrow \bot \mid X_i \in V,\ x_i \in \{\mathbf{t}, \mathbf{f}\} \text{ and } X_i = x_i \text{ is not mentioned in } X = x\}\,.
\end{aligned}
$$

where $Comp(\Pi^{\text{twin}}_{\mathbb{M}} \cup Do(X = x))_\alpha$ is classically equivalent to the following set of formulas (where the connection between atoms of the form V_i^* and Do predicates are eliminated)

$$\{\alpha \ : \ U_i \rightarrow \top \mid U_i \in U\} \cup$$
$$\{\alpha \ : \ V_i \leftarrow F_i \mid V_i = F_i \in F\} \cup$$
$$\{\alpha \ : \ V_i \rightarrow F_i \mid V_i = F_i \in F\} \cup$$
$$\{\alpha \ : \ V_i^* \leftarrow F_i^* \mid V_i \notin X\} \cup$$
$$\{\alpha \ : \ V_i^* \rightarrow F_i^* \mid V_i \notin X\} \cup$$
$$\{\alpha \ : \ V_i^* \leftarrow \top \mid V_i = X_i \text{ for some } X_i \in X \text{ and } x_i = \mathbf{t}\} \cup$$
$$\{\alpha \ : \ V_i^* \rightarrow \top \mid V_i = X_i \text{ for some } X_i \in X \text{ and } x_i = \mathbf{t}\} \cup$$
$$\{\alpha \ : \ V_i^* \rightarrow \bot \mid V_i = X_i \text{ for some } X_i \in X \text{ and } x_i = \mathbf{f}\} \cup$$
$$\{\alpha \ : \ Do(X_i = x_i) \mid X_i \in X, x_i \in x\} \cup$$
$$\{\alpha \ : \ Do(X_i = x_i) \rightarrow \bot \mid X_i \in V,\ x_i \in \{\mathbf{t}, \mathbf{f}\} \text{ and } X_i = x_i \text{ is not mentioned in } X = x\}\,.$$

By Lemma 12, $Comp(\Pi_{\mathbb{M}}^{\text{twin}} \cup Do(X = x))$ is equivalent to the following MLN program

$$\{ln(P(U_i = \mathbf{t})) : \ U_i, ln(P(U_i = \mathbf{f})) : \leftarrow U_i \mid U_i \in U\} \cup$$
$$\{\alpha \ : \ U_i \rightarrow \top \mid U_i \in U\} \cup$$
$$\{\alpha \ : \ V_i \leftarrow F_i \mid V_i = F_i \in F\} \cup$$
$$\{\alpha \ : \ V_i \rightarrow F_i \mid V_i = F_i \in F\} \cup$$
$$\{\alpha \ : \ V_i^* \leftarrow F_i^* \mid V_i \notin X\} \cup$$
$$\{\alpha \ : \ V_i^* \rightarrow F_i^* \mid V_i \notin X\} \cup$$
$$\{\alpha \ : \ V_i^* \leftarrow \top \mid V_i = X_i \text{ for some } X_i \in X \text{ and } x_i = \mathbf{t}\} \cup$$
$$\{\alpha \ : \ V_i^* \rightarrow \top \mid V_i = X_i \text{ for some } X_i \in X \text{ and } x_i = \mathbf{t}\} \cup$$
$$\{\alpha \ : \ V_i^* \rightarrow \bot \mid V_i = X_i \text{ for some } X_i \in X \text{ and } x_i = \mathbf{f}\} \cup$$
$$\{\alpha \ : \ Do(X_i = x_i) \mid X_i \in X, x_i \in x\} \cup$$
$$\{\alpha \ : \ Do(X_i = x_i) \rightarrow \bot \mid X_i \in V,\ x_i \in \{\mathbf{t}, \mathbf{f}\} \text{ and } X_i = x_i \text{ is not mentioned in } X = x\}\,.$$

which can be viewed as the completion of the following LP$^{\text{MLN}}$ program

$$\{ln(P(U_i = \mathbf{t})) : \ U_i, ln(P(U_i = \mathbf{f})) : \leftarrow U_i \mid U_i \in U\} \cup$$

$$\{\alpha \ : \ V_i \leftarrow F_i \mid V_i = F_i \in F\} \cup$$

$$\{\alpha \ : \ V_i^* \leftarrow F_i^*, \neg \ Do(V_i = \mathbf{t}), \neg \ Do(V_i = \mathbf{f}) \mid V_i \notin X\} \cup$$

$$\{\alpha \ : \ V_i^* \mid V_i = X_i \text{ for some } X_i \in X \text{ and } x_i = \mathbf{t}\} \cup$$

$$\{\alpha \ : \ Do(X_i = x_i) \mid X_i \in X, x_i \in x\} \cup$$

$$\{\alpha \ : \leftarrow Do(X_i = x_i) \mid X_i \in V, \ x_i \in \{\mathbf{t}, \mathbf{f}\} \text{ and } X_i = x_i \text{ is not mentioned in } X = x\}$$

which can be further viewed as the corresponding LP$^{\text{MLN}}$ program of the following PCM instance

$$\begin{aligned}
\mathbb{M}^{twin}_{X=x} = \ & \langle\langle U, V \cup V^* \cup \{Do(X_i = x_i) \mid X_i \in X, x_i \in x\}\,, \\
& F \cup F^* \cup \{Do(X_i = x_i) = \top \mid X_i \in X, x_i \in x\} \cup \\
& \{Do(X_i = x_i) = \top \mid X_i \in V, \ x_i \in \{\mathbf{t}, \mathbf{f}\} \text{ and } X_i = x_i \text{ is not mentioned in } X = x\}\rangle, \\
& P(U)\rangle
\end{aligned}$$

where $V^* = \{V_i^* \mid V_i \in V\}$, and

$$F^* = \{V_i^* = F_i^* \mid V_i^* \notin X\} \cup \{X_i^* = x_i \mid X_i^* \in X\}\,.$$

By Proposition 7, we have

$$\begin{aligned}
Pr_{\Pi_{\mathbb{M},X=x} \cup Do(X=x)}[Y^* = y \mid Z = z] \ & = P_{\mathbb{M}^{twin}_{X=x}}(Y^* = y \mid Z = z) \\
& = \frac{\sum_{\left\{u \mid Y^*_{\mathbb{M}^{twin}_{X=x}}(u)=y \ and \ Z_{\mathbb{M}^{twin}_{X=x}}(u)=z\right\}} P(u)}{\sum_{\left\{u \mid Z_{\mathbb{M}^{twin}_{X=x}}(u)=z\right\}} P(u)}
\end{aligned}$$

It can be seen from the definition of $\mathbb{M}^{twin}_{X=x}$ that

$$\left\{u \mid Y^*_{\mathbb{M}^{twin}_{X=x}}(u) = y \ and \ Z_{\mathbb{M}^{twin}_{X=x}}(u) = z\right\} = \{u \mid Y_{\mathbb{M}_{X=x}}(u) = y \ and \ Z_{\mathbb{M}}(u) = z\}$$

and

$$\left\{u \mid Z_{\mathbb{M}^{twin}_{X=x}}(u) = z\right\} = \{u \mid Z_{\mathbb{M}}(u) = z\}\,.$$

So we have

$$
\begin{aligned}
Pr_{\Pi_{\mathbb{M},X=x} \cup Do(X=x)}[Y^* = y \mid Z = z] &= \frac{\sum_{\left\{u \mid Y_{\mathbb{M}_{X=x}}(u)=y \ \ and \ \ Z_{\mathbb{M}}(u)=z\right\}} P(u)}{\sum_{\left\{u \mid Z_{\mathbb{M}}(u)=z\right\}} P(u)} \\
&= P_{\mathbb{M}}(Y_{\mathbb{M}_{X=x}} = y \mid Z = z)
\end{aligned}
$$

□

4.7.7 Proof of Theorem 12

It can be easily seen from the definition of $P(B, r, c = v)$ and the definition of $P(W, c = v)$ that the following two lemmas hold:

Lemma 13. *For any simple P-log program* Π*, any possible world* W *of* Π*, any constant* c *and any* $v \in Dom(c)$ *such that* $c = v$ *is possible in* W*, we have*

$$P(W, c = v) = P(B_{W,c}, r_{W,c}, c = v)$$

Lemma 14. *For any simple P-log program* Π*, any possible world* W *of* Π*, any constant* c *and any* $v \in Dom(c)$ *such that* $c = v$ *is possible in* W*, we have*

$$PR_W(c) = PR_{B_{W,c}, r_{W,c}}(c).$$

Furthermore, the following corollary can be derived:

Corollary 1. *For any simple P-log program* Π*, any possible world* W *of* Π*, any constant* c *and any* $v \in Dom(c)$ *such that* $c = v$ *is possible in* W *and* $W \models c = v$*, we have*

- *If* $PR_W(c) \neq \emptyset$*, then*

$$P(W, c = v) = M_{\Pi^{\mathrm{LP^{MLN}}}}(pf^{c}_{B_{W,c}, r_{W,c}} = v);$$

- *If* $PR_W(c) = \emptyset$*, then*

$$P(W, c = v) = M_{\Pi^{\mathrm{LP^{MLN}}}}(pf^{c}_{\Box, r_{W,c}} = v).$$

For any interpretation I of Π, we define the set $SM_\Pi(I)$ of stable models of $\Pi^{\text{LP}^{\text{MLN}}}$ as follows:

$$SM_\Pi(I) = \left\{ J \mid J \text{ is a (probabilistic) stable model of } \Pi^{\text{LP}^{\text{MLN}}} \text{ such that } J \models F_I \right\}.^{12}$$

The proof of the next lemma uses a restricted version of the splitting theorem in Ferraris *et al.* (2009), which is reformulated as follows:

Theorem 14. *Let* Π_1, Π_2 *be two finite ground programs where rules are of the form (2.1), and* $\mathbf{p}$, $\mathbf{q}$ *be disjoint tuples of distinct atoms. If*

- *Each strongly connected component of the dependency graph of* $\Pi_1 \cup \Pi_2$ *w.r.t.* $\mathbf{p} \cup \mathbf{q}$ *is a subset of* $\mathbf{p}$ *or a subset of* $\mathbf{q}$.
- *No atom in* $\mathbf{p}$ *has a strictly positive occurrence in* Π_2*, and*
- *No atom in* $\mathbf{q}$ *has a strictly positive occurrence in* Π_1.

then an interpretation I *of* $\Pi_1 \cup \Pi_2$ *is a stable model of* $\Pi_1 \cup \Pi_2$ *relative to* $\mathbf{p} \cup \mathbf{q}$ *if and only if* I *is a stable model of* Π_1 *relative to* $\mathbf{p}$ *and* I *is a stable model of* Π_2 *relative to* $\mathbf{q}$.

Lemma 15. *Given a simple P-log program* Π *and a possible world* I *of* Π*, let* $AIRRE_\Pi(I)$ *denote the set of all assignments of the constants in the set*

$$IRRE_\Pi(I) = \sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}}) \setminus \left\{ pf^c_{\Box, r_{I,c}} \mid \substack{c=v,I: \\ c=v \textit{ is possible in } I, \\ I \models c=v \\ \textit{and } PR_I(c)=\emptyset} \right\} \setminus \left\{ pf^c_{B_{I,c}, r_{I,c}} \mid \substack{c=v,I: \\ c=v \textit{ is possible in } I, \\ I \models c=v \\ \textit{and } PR_I(c)\neq\emptyset} \right\}.$$

.

There is a 1-1 correspondence between $SM_\Pi(I)$ *and* $AIRRE_\Pi(I)$.

[12] The formula F_I is defined in Theorem 12.

Proof. We use σ to refer to the signature of $\tau(\Pi)$, and σ' to refer to the signature of $\Pi^{\mathrm{LP}^{\mathrm{MLN}}}$. We construct the 1-1 correspondence as follows.

Given an element J in $SM_\Pi(I)$, i.e., a stable model of $\Pi^{\mathrm{LP}^{\mathrm{MLN}}}$ which satisfies F_I, due to the UEC constraint for constants in $IRRE_\Pi(I)$, $SM_\Pi(I)$ must assign some value to all constants in $IRRE_\Pi(I)$ to be a stable model. We extract the assignment of atoms in $IRRE_\Pi(I)$ from J to obtain the corresponding element in $AIRRE_\Pi(I)$.

Given any arbitrary assignment of constants in $IRRE_\Pi(I)$, we extend this assignment by assigning the constants in $\sigma(\Pi^{\mathrm{LP}^{\mathrm{MLN}}}) \setminus IRRE_\Pi(I)$ in the following way, to obtain the corresponding element J in $SM_\Pi(I)$:

- For all $c = v \in I$, set $c^J = v$.
- For all constants of the form $pf^c_{\Box,r_{I,c}}$, where $c \in \sigma$, $c^I = v$, $c = v$ is possible in I and $PR_I(c) = \emptyset$, set $(pf^c_{\Box,r_{I,c}})^J = v$, and set $(Assigned_{r_{I,c}})^J$ to be undefined.
- For all constants of the form $pf^c_{B_{I,c},r_{I,c}}$, where $c \in \sigma$, $c^I = v$, $c = v$ is possible in I and $PR_I(c) \neq \emptyset$, set $(pf^c_{B_{I,c},r_{I,c}})^J = v$, and set $(Assigned_{r_{I,c}})^J = \mathbf{t}$.

The above construction of J guarantees that J satisfies $\overline{(\Pi^{\mathrm{LP}^{\mathrm{MLN}}})^{hard}}$ and F_I. Next we show that J is a stable model of $\Pi^{\mathrm{LP}^{\mathrm{MLN}}}$:

We split rules in $\overline{\Pi_J^{\mathrm{LP}^{\mathrm{MLN}}}}$ into two subsets $\overline{\Pi_{J,1}^{\mathrm{LP}^{\mathrm{MLN}}}}$ and $\overline{\Pi_{J,2}^{\mathrm{LP}^{\mathrm{MLN}}}}$ as follows:

- $\overline{\Pi_{J,1}^{\mathrm{LP}^{\mathrm{MLN}}}}$ contains all rules in $\tau(\Pi)$, and rules of the following forms:
  1. $c = v \leftarrow B, B', pf^c_{B',r} = v, not\ intervene(c)$, where c is a constant of σ, $v \in Dom(c)$, B is the body of some random selection rule r of the form $[r]\, random(c) \leftarrow B$, and B' appears in some pr-atom of the form $pr(c = v \mid B') = p$ where $p \in [0, 1]$;

2. $c = v \leftarrow B, pf^c_{\Box,r} = v, not\ Assigned_r, not\ intervene(c)$, where c is a constant of σ, $v \in Dom(c)$, and B is the body of some random selection rule r of the form $[r]\, random(c) \leftarrow B$;

- $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,2}}$ is $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J}} \setminus \overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,1}}$

It can be seen that no atom in σ has a strictly positive occurrence in $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,2}}$, and no atom in $\sigma' \setminus \sigma$ (Atoms of the form "$Assigned_r$" and "$pf^c_{\_,r}$") has a strictly positive occurrence in $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,1}}$. Furthermore, the construction of $\Pi^{\mathrm{LP}^{\mathrm{MLN}}}$ guarantees that all loops of size greater than one involves atoms in σ only. So each strongly connected component of the dependency graph of $\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J}$ w.r.t. σ' is a subset of σ or a subset of $\sigma' \setminus \sigma$. By Theorem 14, it suffices to show that J is a stable model of $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,1}}$ relative to σ and J is a stable model of $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,2}}$ relative to $\sigma' \setminus \sigma$.

- **J is a stable model of $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,1}}$ relative to σ:** Since I is a stable model of $\tau(\Pi)$ relative to σ, J is a stable model of $\tau(\Pi)$ relative to σ. It can be easily seen from the construction of J that $J \vDash \overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,1}}$. Since $\tau(\Pi)$ is a subset of $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,1}}$, by Proposition 1, J is a stable model of $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,1}}$ relative to σ.

- **J is a stable model of $\overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,2}}$ relative to $\sigma' \setminus \sigma$:** It can be easily seen from the construction of J that $J \vDash \overline{\Pi^{\mathrm{LP}^{\mathrm{MLN}}}_{J,2}}$. Also as we discussed earlier, all loops of size greater than one do not involve atoms in $\sigma' \setminus \sigma$. So it suffices to show that the loop formula of each loop consisting of a single atom in $\sigma' \setminus \sigma$ is satisfied by J. $\sigma' \setminus \sigma$ contains two types of atoms: 1) atoms of the form $Assigned_r$, where r is some random selection rule, and 2) atoms of the form $pf^c_{\_,r} = v$, where c is a constant of σ, $\_$ is $\Box$ or B such that $pr(c = v' \mid B) = p$ is a pr-atom in Π, $v \in Dom(c)$, and r is a random selection rule of the form $[r]\, random(c) \leftarrow B'$.

  – Consider atoms of the form 1). These atoms appear and only appear at the head of rules of the form

$$Assigned_r \leftarrow B, B', not\ Intervene(c).$$

where c is the atom associated with the random selection rule r, B' is the body of the random selection rule r, and B occurs in some pr-atom $pr(c = v \mid B) = p$. The body of this rule involves atoms in σ only. The construction of J sets $Assigned_r$ to be true only when $PR_I(c) \neq \emptyset$, which implies $Assigned_r$ is true in J only when J satisfies $not\ Intervene(c)$, B and B'. Note that $B, not\ Intervene(c)$ does not contain $Assigned_r$. So clearly $B, B' not\ Intervene(c)$ is a one disjunctive term in $ES_{\Pi^{\mathrm{LP^{MLN}}}}(\{Assigned_r\})$. So $Assigned_r \rightarrow ES_{\Pi^{\mathrm{LP^{MLN}}}}(\{Assigned_r\})$ is satisfied.

– Consider atoms of the form 2). Each of these atoms appears and only appears as an atomic fact in $\overline{\Pi_{J,2}^{\mathrm{LP^{MLN}}}}$. So the loop formulas for these atoms are of the form $pf^{c}_{\_,r} = v \rightarrow \top$. Clearly these formulas are satisfied by J.

So J must be a stable model of $\overline{\Pi_{J,2}^{\mathrm{LP^{MLN}}}}$ relative to $\sigma' \setminus \sigma$.

□

Lemma 16. *For any simple P-log program* Π *and any possible world* I *of* Π*, we have*

$$\hat{\mu}_{\Pi}(I) = \sum_{J : J \in SM_{\Pi}(I)} W''_{\Pi^{\mathrm{LP^{MLN}}}}(J).$$

Proof.

$$
\begin{aligned}
\hat{\mu}_\Pi(I) &= \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I\\ \text{and } I\models c=v}} P(I,c=v)\\
&= \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}} P(I,c=v) \times \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}} P(I,c=v)\\
\text{(Corollary 1)} &= \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{B_{I,c},r_{I,c}}=v) \times \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{\Box,r_{I,c}}=v)\\
&= \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{B_{I,c},r_{I,c}}=v) \times \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{\Box,r_{I,c}}=v)\times\\
&\qquad \prod_{\substack{\text{pf:}\\ pf\in\sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}})\setminus\left\{pf^c_{\Box,r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}\right\}\setminus\\ \left\{pf^c_{B_{I,c},r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}\right\}}} \sum_{v:v\in Dom(pf)} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf=v)
\end{aligned}
$$

Consider interpretations in the set $SM_\Pi(I)$. By Lemma 15, there is a 1-1 correspondence between those interpretations and assignments to constants in the set $\sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}})\setminus\left\{pf^c_{\Box,r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}\right\}\setminus\left\{pf^c_{B_{I,c},r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}\right\}$. Furthermore, for each of those interpretations J, $W''(J)$ is precisely the product of the probability assigned to constants in $\sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}})$. Since the third term of the last equation above ranges over all assignments to constants in the set $\sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}})\setminus$

$$\left\{pf^c_{\Box,r_{I,c}} \mid \substack{c=v, I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}\right\} \setminus \left\{pf^c_{B_{I,c},r_{I,c}} \mid \substack{c=v, I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}\right\}, \text{ we have}$$

$$\begin{aligned}
\hat{\mu}_\Pi(I) = &\prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{B_{I,c},r_{I,c}}=v) \times \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{\Box,r_{I,c}}=v)\times \\
&\sum_{J:J\in SM_\Pi(I)} \prod_{\substack{\text{pf:}\\ pf\,\in\,\sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}})\setminus\left\{pf^c_{\Box,r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}\right\}\setminus\\ \left\{pf^c_{B_{I,c},r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}\right\}}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf=pf^J) \\
= &\sum_{J:J\in SM_\Pi(I)} \Bigg[\prod_{\substack{\text{pf:}\\ pf\,\in\,\sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}})\setminus\left\{pf^c_{\Box,r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}\right\}\setminus\\ \left\{pf^c_{B_{I,c},r_{I,c}} \mid \substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}\right\}}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf=pf^J)\times \\
&\prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)\neq\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{B_{I,c},r_{I,c}}=v) \times \prod_{\substack{c=v,I:\\ c=v \text{ is possible in } I,\\ I\models c=v\\ \text{and } PR_I(c)=\emptyset}} M_{\Pi^{\text{LP}^{\text{MLN}}}}(pf^c_{\Box,r_{I,c}}=v)\Bigg] \\
= &\sum_{J:J\in SM_\Pi(I)} \prod_{c=v\,\in\,\sigma^{pf}(\Pi^{\text{LP}^{\text{MLN}}}) \text{ and } c^J=v} M_{\Pi^{\text{LP}^{\text{MLN}}}}(c=v) \\
= &\sum_{J:J\in SM_\Pi(I)} W''_{\Pi^{\text{LP}^{\text{MLN}}}}(J).
\end{aligned}$$

$\square$

Lemma 17. *Given a consistent simple P-log program Π of signature σ, for every stable model J of $\Pi^{\text{LP}^{\text{MLN}}}$ (whose signature is denoted by σ'), J's restriction on σ is a possible world of Π.*

Proof. We construct J's restriction on σ by defining $c^I = c^J$ for all $c \in \sigma$.

- Clearly $J \in SM_\Pi(I)$.
- Now we show that I is a possible world of Π. Since Π is consistent, $\tau(\Pi)$ is satisfiable, and thus $J \models \tau(\Pi)$ (Otherwise J would not be a stable model of $\Pi^{\text{LP}^{\text{MLN}}}$ according to Proposition 3). Since $J \models \tau(\Pi)$, we get $I \models \tau(\Pi)$. To

see that I is a stable model of $\Pi^{\rm LP^{MLN}}$, we consider the loop formula $L^\wedge \rightarrow ES_{\tau(\Pi)}(L)$ for any loop of L of $\tau(\Pi)$ such that $I \models L^\wedge$. L is a loop of $\overline{\Pi_J^{\rm LP^{MLN}}}$ as well since $J \models \tau(\Pi)$, and it is satisfied by J since $I \subseteq J$. Since J is a stable model of $\overline{\Pi_J^{\rm LP^{MLN}}}$, we have

$$J \models ES_{\overline{\Pi_J^{\rm LP^{MLN}}}}(L),$$

i.e.,

$$J \models \bigvee_{\substack{A\cap L\neq\emptyset \\ A\leftarrow B\wedge N\in\overline{\Pi_J^{\rm LP^{MLN}}} \\ B\cap L=\emptyset}} (B \wedge N \wedge \bigwedge_{b\in A\setminus L} \neg b).$$

Consider the following two cases:

- L contains only atoms that are not possible in I. Since those atoms do not occur in the head of any rules in $\overline{\Pi^{\rm LP^{MLN}}} \setminus \tau(\Pi)$, those rules do not contribute in $ES_{\overline{\Pi_J^{\rm LP^{MLN}}}}(L)$. So $ES_{\overline{\Pi_J^{\rm LP^{MLN}}}}(L) = ES_{\tau(\Pi)}(L)$ in this case. Since $\tau(\Pi)$ involves atoms in σ only, and I and J agree on atoms in σ, we have

$$I \models ES_{\tau(\Pi)}(L).$$

- L contains some atoms that are possible in I. In this case, since $J \models ES_{\overline{\Pi_J^{\rm LP^{MLN}}}}(L)$, there must be at least one rule $A \leftarrow B \wedge N \in \overline{\Pi_J^{\rm LP^{MLN}}}$ such that $A \cap L \neq \emptyset$, $B \cap L = \emptyset$ and $J \models B \wedge N \wedge \bigwedge_{b\in A\setminus L} \neg b$. There are again two possible cases:

  * $A \leftarrow B \wedge N \in \tau(\Pi)$. In this case, since $\tau(\Pi)$ involves atoms in σ only, and I and J agree on atoms in σ, we have $I \models B \wedge N \wedge \bigwedge_{b\in A\setminus L} \neg b$. Since this rule contributes to $ES_{\tau(\Pi)}(L)$ as well, we have $I \models ES_{\tau(\Pi)}(L)$.

* $A \leftarrow B \wedge N \notin \tau(\Pi)$. According to the construction of $\Pi^{\text{LP}^{\text{MLN}}}$, $A \leftarrow B \wedge N$ must be of one of the following two forms:

$$c = v \leftarrow B', pf^{c}_{\Box,r} = v, not\ Assigned_r$$

or

$$c = v \leftarrow B'', B', pf^{c}_{B,r} = v, not\ Intervene(c)$$

where $c = v$ is some atom possible in I, r is the random selection rule of the form

$$[r]\ random(c) \leftarrow B',$$

and B'' is the body of some pr-atom related to c and r. In either case, J satisfies B', which involves atoms in σ only. So I satisfies B' as well. Consider the following rule in $\tau(\Pi)$:

$$c = v_1; \ldots; c = v_n \leftarrow B', not\ Intervene(c). \tag{4.17}$$

There are two possible cases:

· J does not satisfy $Intervene(c)$. In this case, (4.17) is satisfied by J, and clearly

$$c = v_1; \ldots; c = v_n \leftarrow B', not\ Intervene(c) \in \left\{ A \leftarrow B \wedge N \mid \substack{A \cap L \neq \emptyset \\ A \leftarrow B \wedge N \in \tau(\Pi) \\ B \cap L = \emptyset} \right\}.$$

So one disjunctive term of $ES_{\tau(\Pi)}(L)$ is satisfied by I. So $ES_{\tau(\Pi)}(L)$ is satisfied by I.

· J satisfies $Intervene(c)$. In this case, for J to be a stable model of $\Pi^{\text{LP}^{\text{MLN}}}$, there must be a rule of the following form

$$Intervene(c) \leftarrow Do(c = v)$$

in $\tau(\Pi)$, where $c = v \in J$ and $c = v \in I$, whose body is satisfied by J, which means the following rule

$$c = v \leftarrow Do(c = v)$$

in $\tau(\Pi)$ is satisfied by J. Clearly

$$c = v \leftarrow Do(c = v) \in \left\{ A \leftarrow B \wedge N \mid {\scriptstyle \substack{A \cap L \neq \emptyset \\ A \leftarrow B \wedge N \in \tau(\Pi) \\ B \cap L = \emptyset}} \right\}.$$

So one disjunctive term of $ES_{\tau(\Pi)}(L)$ is satisfied by I. So $ES_{\tau(\Pi)}(L)$ is satisfied by I.

So I satisfies $ES_{\tau(\Pi)}(L)$ for all loops L of $\tau(\Pi)$. Consequently, I is a stable model of $\tau(\Pi)$, and thus I is a possible world of Π.

So I is a stable model of $\tau(\Pi)$, and thus a possible world of Π.

$\square$

Theorem 12 *For any consistent simple P-log program Π of signature σ and any possible world W of Π, we construct a formula F_W as follows.*

$$\begin{aligned} F_W = \quad & (\textstyle\bigwedge_{c=v \in W} c = v) \wedge \\ & (\bigwedge_{\substack{c,v: \\ c=v \text{ is possible in } W, \\ W \models c=v \text{ and } PR_W(c) \neq \emptyset}} pf^c_{B_{I,c}, r_{I,c}} = v) \\ & \wedge (\bigwedge_{\substack{c,v: \\ c=v \text{ is possible in } W, \\ W \models c=v \text{ and } PR_W(c) = \emptyset}} pf^c_{\Box, r_{I,c}} = v) \end{aligned}$$

We have

$$\mu_\Pi(W) = P_{\Pi^{\mathrm{LP^{MLN}}}}(F_W).$$

For any proposition A of signature σ,

$$P_\Pi(A) = P_{\Pi^{\mathrm{LP^{MLN}}}}(A).$$

Proof. We first show

$$\sum_{W \text{ is a possible world of } \Pi} \hat{\mu}_{\Pi}(W) \qquad = \qquad \sum_{J \in SM''\left[\Pi^{\text{LP}^{\text{MLN}}}\right]} W''_{\Pi^{\text{LP}^{\text{MLN}}}}(J)$$

i.e., the normalization factor of $\hat{\mu}$ is the normalization factor of $W''_{\Pi^{\text{LP}^{\text{MLN}}}}$.

By Lemma 16 we have,

$$\sum_{W \text{ is a possible world of } \Pi} \hat{\mu}_{\Pi}(W) \qquad = \qquad \sum_{W \text{ is a possible world of } \Pi} \; \sum_{J \in SM_{\Pi}(W)} W''_{\Pi^{\text{LP}^{\text{MLN}}}}(J) \tag{4.18}$$

By Lemma 17, for every stable model J of $\Pi^{\text{LP}^{\text{MLN}}}$, there exists a possible world W of Π such that $J \in SM_{\Pi}(W)$. So we can enumerate all stable models of $\Pi^{\text{LP}^{\text{MLN}}}$ by enumerating all possible worlds W of Π and enumerating all elements in $SM_{\Pi}(W)$ for each W, and thus the right-hand side of (4.18) can be rewritten as

$$\sum_{J \text{ is a stable model of } \Pi^{\text{LP}^{\text{MLN}}}} W''_{\Pi^{\text{LP}^{\text{MLN}}}}(J).$$

By Lemma 2, an interpretation J is a stable model of $\Pi^{\text{LP}^{\text{MLN}}}$ if and only if $J \in SM''\left[\Pi^{\text{LP}^{\text{MLN}}}\right]$. So the right-hand side of (4.18) can be further rewritten as

$$\sum_{J \in SM''\left[\Pi^{\text{LP}^{\text{MLN}}}\right]} W''_{\Pi^{\text{LP}^{\text{MLN}}}}(J).$$

Thus we have

$$
\begin{aligned}
\mu_\Pi(W) &= \frac{\hat{\mu}_\Pi(W)}{\sum_{W \text{ is a possible world of } \Pi} \hat{\mu}(W)} \\
&= \frac{\hat{\mu}_\Pi(W)}{\sum_{J \in SM''\left[\Pi^{\mathrm{LP^{MLN}}}\right]} W''_{\Pi^{\mathrm{LP^{MLN}}}}(J)} \\
\text{(By Lemma 16)} &= \frac{\sum_{J \in SM_\Pi[W]} W''_{\Pi^{\mathrm{LP^{MLN}}}}(J)}{\sum_{J \in SM''\left[\Pi^{\mathrm{LP^{MLN}}}\right]} W''_{\Pi^{\mathrm{LP^{MLN}}}}(J)} \\
&= \sum_{J \in SM_\Pi[W]} \frac{W''_{\Pi^{\mathrm{LP^{MLN}}}}(J)}{\sum_{J \in SM''\left[\Pi^{\mathrm{LP^{MLN}}}\right]} W''_{\Pi^{\mathrm{LP^{MLN}}}}(J)} \\
&= \sum_{J \in SM_\Pi[W]} P''_{\Pi^{\mathrm{LP^{MLN}}}}(J)
\end{aligned}
$$

For those interpretations J that do not belong to $SM_\Pi[W]$ but satisfy F_W, it must be the case that J is not a stable model of $\Pi^{\mathrm{LP^{MLN}}}$. By Lemma 2, $P''_{\Pi^{\mathrm{LP^{MLN}}}}(J) = 0$. So we have

$$
\begin{aligned}
\mu_\Pi(W) &= \sum_{J \in SM_\Pi[W] \text{ and } J \models F_W} P''_{\Pi^{\mathrm{LP^{MLN}}}}(J) + \sum_{J \notin SM_\Pi[W] \text{ and } J \models F_W} P''_{\Pi^{\mathrm{LP^{MLN}}}}(J) \\
&= \sum_{J \models F_W} P''_{\Pi^{\mathrm{LP^{MLN}}}}(J) \qquad (4.19)
\end{aligned}
$$

and consequently by Theorem 4,

$$
\begin{aligned}
\mu_\Pi(W) &= \sum_{J \models F_W} P_{\Pi^{\mathrm{LP^{MLN}}}}(J) \\
&= P_{\Pi^{\mathrm{LP^{MLN}}}}(F_W). \qquad (4.20)
\end{aligned}
$$

According to the definition,

$$
P_\Pi(F) = \sum_{W \text{ is a possible world of } \Pi \text{ that satisfies } F} \mu_\Pi(W).
$$

Using the above result (4.20), we have

$$P_{\Pi}(F) = \sum_{W \text{ is a possible world of } \Pi \text{ that satisfies } F} P_{\Pi^{\text{LP}^{\text{MLN}}}}(F_W)$$

$$= \sum_{W \text{ is a possible world of } \Pi \text{ that satisfies } F} \; \sum_{W \in SM_{\Pi}(W)} P_{\Pi^{\text{LP}^{\text{MLN}}}}(W).$$

The right-hand side of the last equation is the sum of the probabilities of a collection of stable models of $\Pi^{\text{LP}^{\text{MLN}}}$. Clearly all those stable models of $\Pi^{\text{LP}^{\text{MLN}}}$ satisfies F since they are all from some $SM_{\Pi}(W)$ for some possible world W of Π that satisfies F. Furthermore, given any stable model J of $\Pi^{\text{LP}^{\text{MLN}}}$ that satisfies F, by lemma 17, there exists a possible world W of Π such that $J \in SM_{\Pi}(W)$. Since W and J agree on all atoms in $\sigma(\Pi)$ and $J \models F$, $W \models F$. So the probability of J is counted in the right-hand side of the above equation. Finally, obviously no two stable models of $\Pi^{\text{LP}^{\text{MLN}}}$ are counted twice. Hence, the right-hand side can be rewritten as

$$P_{\Pi^{\text{LP}^{\text{MLN}}}}(F),$$

and thus we have

$$P_{\Pi}(F) = P_{\Pi^{\text{LP}^{\text{MLN}}}}(F).$$

$\square$

Chapter 5

LP$^{\text{MLN}}$ INFERENCE

As we mentioned in Chapter 4, LP$^{\text{MLN}}$ can be embedded in ASP with weak constraints and Markov Logic. Following this result, we have implemented two systems to compute LP$^{\text{MLN}}$. our systems, LPMLN2ASP 1.0 and LPMLN2MLN 1.0, compute LP$^{\text{MLN}}$ by translating LP$^{\text{MLN}}$ programs into ASP programs and MLN programs, resp., In this chapter, we will go over each of these two systems.

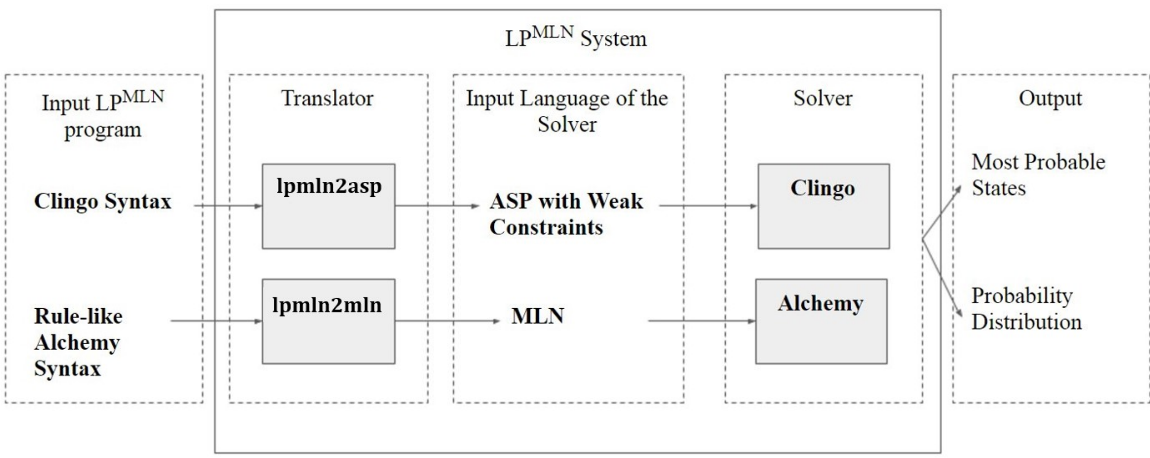


Figure 5.1: LP$^{\text{MLN}}$ System Architecture

Figure 5.1 shows the overview of the implementations. Each of the input languages of LPMLN2ASP 1.0 and LPMLN2ASP 1.0 adopts the syntax of the target language that it is translated into. More precisely, the input language of LPMLN2ASP 1.0 is identical to the input language of CLINGO except that weights are prepended to soft rules. The input language of LPMLN2MLN 1.0 adopts the syntax of input language of ALCHEMY with minor modifications, such as using `<=` instead of `=>`. This is intended for the users who are already experienced with CLINGO and ALCHEMY.

The systems are publicly available at `http://reasoning.eas.asu.edu/lpmln/`,

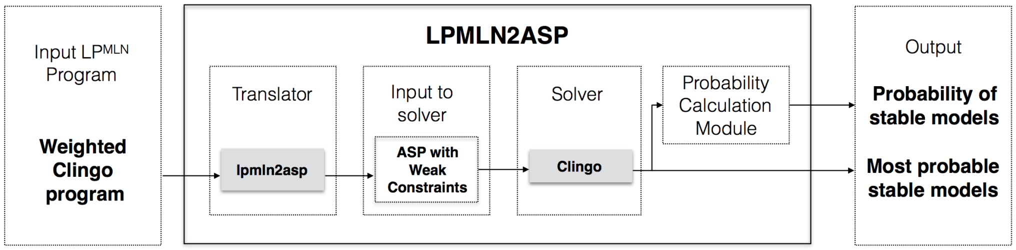


Figure 5.2: Architecture of System LPMLN2ASP 1.0

along with the user manual and examples. We refer the reader to the system homepage for more details.

5.1 System LPMLN2ASP 1.0

System LPMLN2ASP 1.0 is an implementation of LP$^{\text{MLN}}$ based on the result in Section 4.1.3 using CLINGO v4.5. It can be used for computing the probabilities of stable models, marginal/conditional probability of a query, as well as the most probable stable models.

In the input language of LPMLN2ASP 1.0, a soft rule is written in the form

$$w_i \quad Head_i \leftarrow Body_i \tag{5.1}$$

where w_i is a real number in decimal notation, and $Head_i \leftarrow Body_i$ is a CLINGO rule. A hard rule is written without weights and is identical to a CLINGO rule. For instance, the "Bird" example from Section 3.2 can be represented in the input language of LPMLN2ASP 1.0 as follows. The first three rules represent definite knowledge while the last two rules represent uncertain knowledge with different confidence levels.

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

```
1 migratorybird(jo).
```

The basic command line syntax of executing LPMLN2ASP 1.0 is

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

which follows the ALCHEMY command line syntax.

The mode of computation is determined by the options provided to LPMLN2ASP 1.0. By default, the system finds a most probable stable model of $\mathsf{lpmln2asp}^{\rm pnt}(\Pi)$ (MAP estimate) by leveraging CLINGO's built-in optimization method for weak constraints.

For computing marginal probability, LPMLN2ASP 1.0 utilizes CLINGO's interface with Python. When CLINGO enumerates each stable model of $\mathsf{lpmln2asp}^{\rm pnt}(\Pi)$, the computation is interrupted by the *probability computation module*, a Python program which records the stable model as well as its penalty specified in the unsat atoms true in the stable model. Once all the stable models are generated, the control returns to the module, which sums up the recorded penalties to compute the normalization constant as well as the probability of each stable model. The probabilities of query atoms (specified by the option -q) are also calculated by adding the probabilities of the stable models that contain the query atoms. For instance, the probability of a query atom residentbird(jo) is $\sum_{I \models \texttt{residentbird(jo)}} P(I)$. The option -all instructs the system to display all stable models and their probabilities.

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

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

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

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

By default, LPMLN2ASP 1.0 computes MAP inference. For example, `lpmln2asp -i bird.lpmln` returns

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

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

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

For example, `lpmln2asp -i bird.lpmln -all` outputs

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

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

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

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

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

```
residentbird(jo) 0.665240955775
```

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

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

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

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

where `evid.db` contains

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

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

```
residentbird(jo) 0.73105857863
```

Debugging ASP Programs: The command line to use is

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

By default, LPMLN2ASP 1.0 does not translate hard rules and pass them to CLINGO as is. The option `-hr` instructs the system to translate hard rules as well. According to Proposition 2 by Lee and Wang (2016), as long as the $\mathrm{LP}^{\mathrm{MLN}}$ program has a probabilistic stable model that satisfies all hard rules, the simpler translation that does not translate hard rules gives the same result as the full translation and is more computationally efficient. Since in many cases hard rules represent definite knowledge that should not be violated, this is desirable.

On the other hand, translating hard rules could be relevant in some other cases, such as debugging an answer set program by finding which rules cause inconsistency. For example, consider a CLINGO input program `bird.lp`, that is similar to `bird.lpmln` but drops the weights in the last two rules. CLINGO finds no stable models for this program. However, if we invoke LPMLN2ASP 1.0 on the same program as

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

the output of LPMLN2ASP 1.0 shows three probabilistic stable models, each of which shows a way to resolve the inconsistency by ignoring the minimal number of the rules. For instance, one of them is {`bird(jo)`, `residentbird(jo)`}, which disregards the last rule. The other two are similar.

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

5.1.1 Computing MLN with LPMLN2ASP *1.0*

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

$$\begin{array}{lll} w: \quad smoke(x) \wedge influence(x,y) \rightarrow smoke(y) & & \\ \alpha: smoke(alice) & \alpha: influence(alice, bob) & \alpha: influence(bob, carol). \end{array} \tag{5.2}$$

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

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

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

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

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

The result can be verified by LPMLN2ASP 1.0. For $w = 1$, the input program `smoke.lpmln` is

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

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

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

as expected.

On the other hand, if (5.2) is understood under the MLN semantics (assuming `influence` relation is fixed as before), similar to above, one can compute

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

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

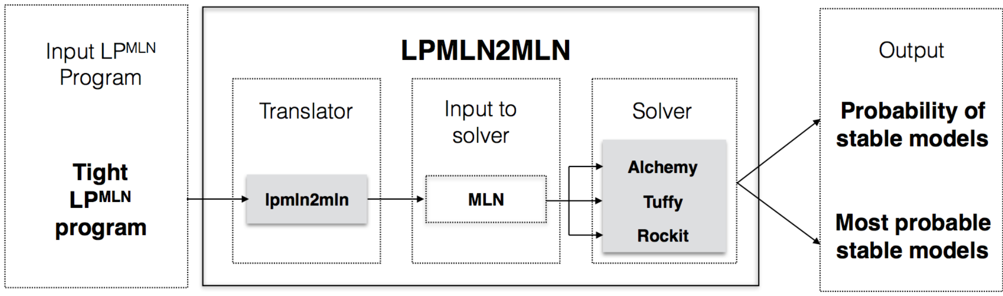


Figure 5.3: Architecture of System LPMLN2MLN 1.0

According to Theorem 6, MLN can be easily embedded in LP$^{\text{MLN}}$ by adding a choice rule for each atom with an arbitrary weight, similar to the way propositional logic can be embedded in ASP using choice rules. Consequently, it is possible to use system LPMLN2ASP 1.0 to compute MLN, which is essentially using an ASP solver to compute MLN.

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

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

which agrees with the computation above.

5.2 System LPMLN2MLN 1.0

System LPMLN2MLN 1.0 is an implementation of LP$^{\text{MLN}}$ based on the result in Section 4.2.2 using ALCHEMY (v2.0), TUFFY (v0.3) and ROCKIT (v0.5).

The basic command line syntax of executing LPMLN2MLN 1.0 is

```
lpmln2mln -i <input file> -r <output file> -q <query predicates>
     [-e <evidence file>]
     [-tuffy| -rockit| -alchemy] [-mln "<options for mln solvers>"]
```

which is similar to the command of executing LPMLN2ASP 1.0.

The syntax of the input language of LPMLN2MLN 1.0 follows that of ALCHEMY, except that it uses a rule form. For example, consider again "Bird" example in Section 3.2. In the input language of LPMLN2MLN 1.0, it is encoded as

```
entity={Jo}

Bird(entity)
MigratoryBird(entity)
ResidentBird(entity)
```

```
Bird(x) <= ResidentBird(x).
Bird(x) <= MigratoryBird(x).
<= ResidentBird(x) ^ MigratoryBird(x).

2 ResidentBird(Jo)
1 MigratoryBird(Jo)
```

Executing

```
lpmln2mln -i bird.lpmln -r out -q Bird,ResidentBird,MigratoryBird
```

gives

```
Bird(Jo) 0.90296   ResidentBird(Jo) 0.667983   MigratoryBird(Jo) 0.235026
```

(When no MLN solver is specified in the command line, ALCHEMY is called by default.)

5.3 Comparison between Two LP$^{\text{MLN}}$ Implementations

When the domain is small, our experience is that it is much more convenient to work with LPMLN2ASP 1.0 because it supports many useful ASP constructs and its exact computation yields outputs that are easier to understand. Once we make sure the program is correct and we do not need advanced ASP constructs nor recursive definitions, we may use LPMLN2MLN 1.0 for more scalable inference.

We report the running time statistics for both LPMLN2ASP 1.0 and LPMLN2MLN 1.0 on the example of finding a maximal "relaxed clique" in a graph, where the goal is to select as many nodes as possible while a penalty is assigned for each pair of disconnected nodes. The penalty assigned to disconnected nodes and the reward given to each node included in the subgraph define how much "relaxed" the clique is.

The LPMLN2ASP 1.0 encoding of the relaxed clique example is

```
{in(X)} :- node(X).
disconnected(X, Y) :- in(X), in(Y), not edge(X, Y).
5  :- not in(X), node(X).
5  :- disconnected(X, Y).
```

The LPMLN2MLN 1.0 encoding of the relaxed clique example is

```
{In(x)} <= Node(x).
Disconnected(x, y) <= In(x) ^ In(y) ^ !Edge(x, y).
5  <= !In(x) ^ Node(x)
5  <= Disconnected(x, y)
```

We use a Python script to generate random graphs with each edge generated with a fixed probability p. We experiment with $p = 0.5, 0.8, 0.9, 1$ and different numbers of nodes. For each problem instance, we perform MAP inference to find a

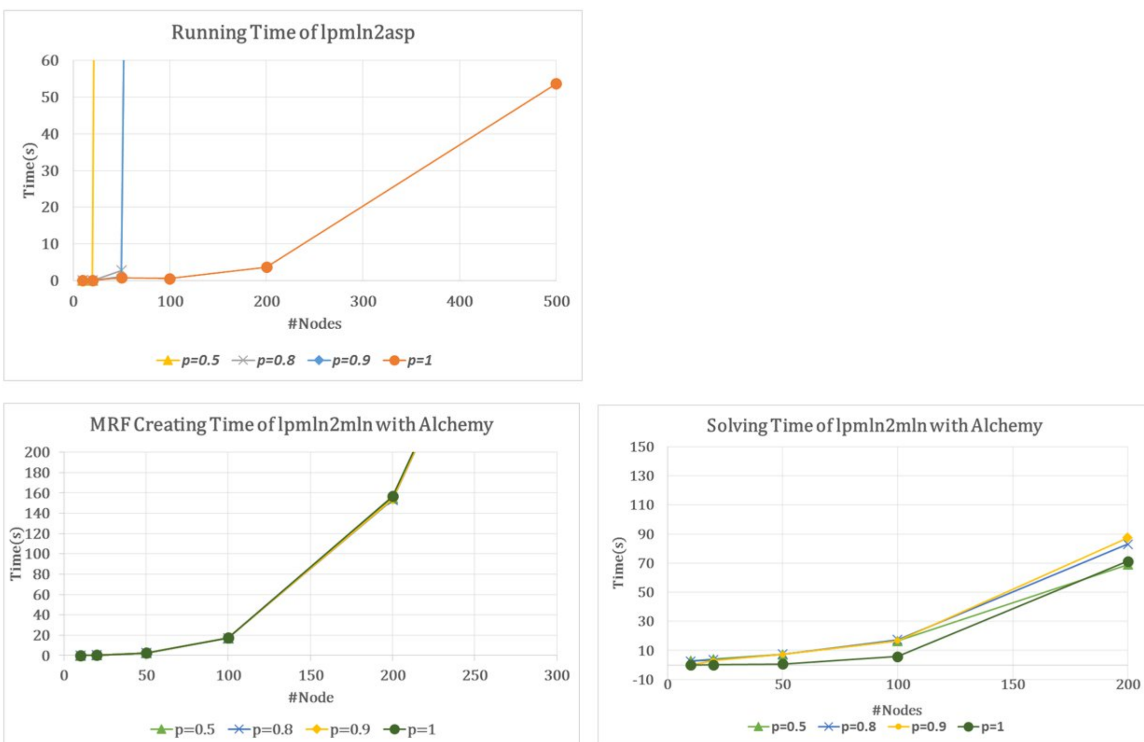


Figure 5.4: Running Statistics on Finding Relaxed Clique

maximal relaxed clique with both LPMLN2ASP 1.0 and LPMLN2MLN 1.0. The timeout is 20 minutes. The experiments were performed on a machine powered by 4 Intel(R) Core(TM) i5-2400 CPU with OS Ubuntu 14.04.5 LTS and 8G memory.

Figure 5.4 shows running statistics of utilizing different underlying solvers. For LPMLN2ASP 1.0, grounding finishes almost instantly for all problem instances that we tested. We plot how solving times vary according to the number of nodes for different edge generation probabilities (top left graph). Roughly, solving time increases as the number of nodes increases. However, there is no clear correlation between solving time and the edge probability (i.e., the density of the graph). For $p = 0.5$, the LPMLN2ASP 1.0 system first times out when $\#Nodes = 50$, while for both $p = 0.8$ and $p = 0.9$, it first times out when $\#Node = 100$. On the other hand, when $\#Node = 20$, solving time roughly increases as the edge probability increases except for $p = 0.5$. The running time is sensitive to particular problem instances, due to the exact optimization algorithm CDNL-OPT Gebser *et al.* (2011) used by CLINGO, which only terminates when a true optimal solution is found. The non-deterministic nature of CDNL-OPT also brings randomness on the path through which an optimal solution is found, which makes the running time differ even among similar-sized problem instances, while in general, as the size of the graph increases, the search space gets larger, thus the solving time increases.

For LPMLN2MLN 1.0 with ALCHEMY (bottom left and bottom right), grounding (MRF creating time) becomes the bottleneck. It increases much faster than solving time, and times out first when $\#Nodes = 500$. Again, the running time increases as the number of nodes increases. On the other hand, unlike LPMLN2ASP 1.0, ALCHEMY uses MaxWalkSAT for MAP inference, which allows a suboptimal solution to be returned. The approximate nature of the method allows relatively consistent running times for different problem instances, as long as parameters such as the maximum number of iterations/tries are fixed among all experiments. The running times are not also much affected by the edge probability.

In general, LPMLN2MLN 1.0 can be more scalable via parameter setting, while LPMLN2ASP 1.0 grants better solution quality. LPMLN2MLN 1.0 with TUFFY shows a similar behavior as LPMLN2MLN 1.0 with ALCHEMY.

5.4 Using LP$^{\text{MLN}}$ Systems to Compute Other Languages

5.4.1 Computing ProbLog

As discussed in Section 4.3, ProbLog can be viewed as a special case of the LP$^{\text{MLN}}$ language, in which soft rules are atomic facts only. System PROBLOG2 implements a native inference and learning algorithm which converts probabilistic inference problems into weighted model counting problems and then solves with knowledge compilation methods Fierens *et al.* (2013). We compare the performance of LPMLN2ASP 1.0 with that of PROBLOG2 on ProbLog input programs. We encode the problem of reachability in a probabilistic graph in both languages, and perform MAP inference ("given that there is a path between two nodes, what is the most likely graph?") as well as marginal probability computation ("given two particular nodes, what is the probability that there exists a path between them?"). We use a Python script to generate edges with probabilities randomly assigned. For the probabilistic facts $p :: \mathtt{edge}(n_1, n_2)$ $(0 < p < 1)$ in PROBLOG2, we write $ln(p/(1-p)) : \mathtt{edge}(n_1, n_2)$ for LPMLN2ASP 1.0,

which makes the probability of the edge being true to be p and being false to be $1 - p$.

The path relation is defined in the input language of LPMLN2ASP 1.0 as

```
path(X,Y) :- edge(X,Y).
```

Parameter	MAP	
	LPMLN2ASP (CLINGO 4.5)	PROBLOG2
$\#edges = 9$	0.013s	0.192s
$\#edges = 25$	0.021s	Timeout
$\#edges = 81$	0.308s	Timeout
$\#edges = 100$	0.756s	Timeout
$\#edges = 225$	6.121s	Timeout
$\#edges = 400$	29.706s	Timeout

Parameter	Marginal		
	LPMLN2ASP (CLINGO 4.5)	PROBLOG2 Default Setting	PROBLOG2 Sample Based $\#_{Sample} = 1000$
$\#edges = 9$	0.520s	0.137s	1.878s
$\#edges = 10$	0.676s	1.468s	1.656s
$\#edges = 11$	1.396s	1.480s	1.684s
$\#edges = 12$	2.524s	1.781s	1.672s
$\#edges = 13$	4.995s	Timeout	1.692s
$\#edges = 14$	9.744s	Timeout	1.796s
$\#edges = 15$	19.568s	Timeout	1.732s
$\#edges = 16$	38.476s	Timeout	1.748s
$\#edges = 18$	164.192s	Timeout	16.676s
$\#edges = 19$	306.000s	Timeout	4.564s
$\#edges = 20$	637.584s	Timeout	4.020s
$\#edges = 21$	Timeout	Timeout	4.344s

Figure 5.5: Running Statistics on Reachability in a Probabilistic Graph

```
path(X,Y) :- path(X,Z), path(Z, Y), Y != Z.
```

and in the input language of PROBLOG2 as

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- path(X,Z), path(Z,Y), Y \== Z.
```

Figure 5.5 shows the running time of each experiment. LPMLN2ASP 1.0 outperforms PROBLOG2 with the default setting (exact inference) in both MAP inference and marginal probability computation. However, both systems' marginal probability computations are not scalable because they enumerate all models. Using a sampling-based inference instead, PROBLOG2 is able to handle marginal probability computation more effectively (the MAP inference in PROBLOG2 is exact inference only). In general, compared to running on tight programs, PROBLOG2 is slow for non-tight programs such as the program we use here. A possible reason is that it has to convert the input program, combined with the query, into weighted Boolean formulas, which is expensive for non-tight programs.

5.4.2 Reasoning about Probabilistic Causal Model

Section 4.4.2 has shown how to represent Pearl's probabilistic causal model by LP$^{\text{MLN}}$. Due to the acyclicity assumption on the causality, the LP$^{\text{MLN}}$ representation is tight, so we can use either implementation of LP$^{\text{MLN}}$ to compute probabilistic queries on a PCM. (Related to this, Appendix A of Lee *et al.* (2017) shows how Bayesian networks can be represented in LP$^{\text{MLN}}$.)

Consider Example 5 again, We illustrate how we use LP$^{\text{MLN}}$ systems to compute the counterfactual query "Given that the prisoner is dead, what is the probability that the prisoner would be alive if Rifleman A had not shot?" According to Pearl (2000), the answer is $\frac{(1-p)q}{1-(1-p)(1-q)}$.

Theorem 4 from the paper by Lee *et al.* (2015) states that the counterfactual reasoning in PCM can be reduced to LP$^{\text{MLN}}$ computation. The translation of PCM into LP$^{\text{MLN}}$ in Section 4.4 by Lee *et al.* (2015) can be represented in the input language of LPMLN2ASP 1.0 as follows, where `as`, `bs`, `cs`, `ds` are nodes in the twin network, `a1` means that `a` is true; `a0` means that `a` is false; other atoms are defined similarly. Let $p = 0.7$ and $q = 0.2$.

```
@log(0.7/0.3)  u.
@log(0.2/0.8)  w.

c :- u.

cs :- u, not do(c1), not do(c0).
as :- cs, not do(a1), not do(a0).
as :- w, not do(a1), not do(a0).
```

```
a :- c.
a :- w.
b :- c.
d :- a.
d :- b.

bs :- cs, not do(b1), not do(b0).
ds :- as, not do(d1), not do(d0).
ds :- bs, not do(d1), not do(d0).
```

```
cs :- do(c1).                    bs :- do(b1).
as :- do(a1).                    ds :- do(d1).
```

To represent the counterfactual query, the evidence file contains:

```
do(a0).
:- not d.
```

Note the different ways that intervention (`do(a0)`) and observation (`d`) are encoded.

With the command `lpmln2asp -i pcm.lp -r out -e evid.db -q ds` we obtain `ds 0.921047297896`, which means there is a 8% chance that the prisoner would be alive.

The other queries mentioned in Example 6, i.e., prediction, abduction, transduction, action, can be similarly automated with LPMLN2ASP 1.0.

5.5 Related Work

Computing marginal/conditional probabilities typically involves summing up variable assignments that satisfy the query. While enumerating all variable assignments is an intractable task, there are various ways to make this process more efficient. Fierens *et al.* (2013) computes marginal and conditional probabilities under Problog programs by converting the program together with the query into weighted Boolean formulas and then turn the inference problem into weighted model counting problems, which are then solved by knowledge compilation techniques. In particular, Fierens *et al.* (2013) compiles Problog programs into deterministic-Decomposable Negation Normal Form (d-DNNF) circuits. Vlasselaer *et al.* (2014), instead, compiles Problog programs into Sentential Decision Diagrams (SDD), which yields better performance in inference tasks.

Bellodi *et al.* (2014) have lifted the inference method for probabilistic graphical model,variable elimination, to first-order level, thus able to utilize the underlying regularity of the ground programs, resulting in inference processes whose time complexity does not depend on the size of ground instances. Similarly, Singla and Domingos (2008) introduced lifted belief propagation for Markov Logic and Mihalkova and Richardson (2009) exploit frequent repeated structures and cluster similar query literals to speed up inference. There are also approximate inference algorithms which get around enumerating all possible variable assignments by sampling a representative subset of variable assignments. Various sampling methods have been developed for inference on Markov logic networks. Theoretically, MCMC sampling can be designed for sampling possible worlds of a Markov logic network, such as Gibbs sampling-Domingos and Lowd (2009). However, performance of those algorithms is generally not great in the presence of deterministic or near-deterministic dependencies. To address this problem, Poon and Domingos (2006) proposed the sampling algorithm MC-SAT, which is a slice MCMC algorithm that combines MCMC sampling with satisfiability checking. Possible worlds are sampled from slices, which are determined by subprograms sampled from the whole program. In Chapter 6, we will present a sampling algorithm for sampling $\mathrm{LP}^{\mathrm{MLN}}$ stable model, called MC-ASP, adapted from MC-SAT. MC-ASP can be easily used to perform sampling-based inference for $\mathrm{LP}^{\mathrm{MLN}}$.

Shterionov *et al.* (2010) computes the probability of queries by sampling assignments on probabilistic atoms in a hierarchical way, with the hierarchy obtained from DNF formulas converted from the program together with the query.

For Most Probable Explanation (MPE) inference, one widely used algorithm is MaxWalkSAT (3.1, Domingos and Lowd (2009)). MaxWalkSAT repeatedly picking an unsatisfied clause at random and flipping the truth value of one atom in the clause.

With a predefined probability, the choice on the atom alternate between 1) a random atom, and 2) the atom that maximize the weighted sum of satisfied clauses, if flipped. This alternation prevent the search from beings stuck at local optimum.

Nickles (2018) have established sampling methods for PrASP (Nickles and Mileo (2014)) that can be used for inference. As mentioned in Section 4.6, in PrASP, the weights of rules are explicitly probabilities. The sampling methods need to either first explicitly find out a distribution over possible worlds that satisfies the probabilities of rules, and then sample according to the distribution, or generate samples directly but in a way that implicitly reflect the probability distribution. Nickles (2018) presents MCMC based sampling methods for the first approach and simulated annealing and a modified CDNL (Conflict-Driven Nogood Learning; Gebser *et al.* (2012)) algorithm for the second approach.

Chapter 6

LP$^{\text{MLN}}$ WEIGHT LEARNING

In all the above example LP$^{\text{MLN}}$ programs, rule weights are manually specified by the user. This can be done for simple programs, however a systematic assignment of weights for a complex program could be challenging. A desirable way to address this problem is to learn the weights from the observed data. In this section, we discuss weight learning algorithms for LP$^{\text{MLN}}$.

Weight learning in LP$^{\text{MLN}}$ is formulated as to find the weights of the rules in the LP$^{\text{MLN}}$ program such that the likelihood of the observed data according to the LP$^{\text{MLN}}$ semantics is maximized. In LP$^{\text{MLN}}$, due to the requirement of a stable model, deterministic dependencies are often. Poon and Domingos [2006] noted that deterministic dependencies break the support of a probability distribution into disconnected regions, making it difficult to design ergodic Markov chains for MCMC inference, which motivated them to develop a new algorithm called MC-SAT. We adapt that algorithm to LP$^{\text{MLN}}$, which we call MC-ASP. Unlike MC-SAT, MC-ASP utilizes ASP solvers for MCMC sampling. Learning in LP$^{\text{MLN}}$ is in accordance with the stable model semantics, so the learned programs can be used for probabilistic extensions of knowledge-rich domains that involve reachability analysis and reasoning about dynamic domains, for which Markov Logic is not readily applicable.

Throughout this section, we consider only LP$^{\text{MLN}}$ programs whose stable models do not violate hard rules, i.e, LP$^{\text{MLN}}$ programs Π such that $SM'[\Pi]$ is not empty. We further reformulate (3.5) in a penalty-based style, and allowing the program Π to be non-ground:

$$W'_{\Pi}(I) = \begin{cases} exp\Bigg(- \sum\limits_{w_i:R_i \,\in\, \Pi^{\text{soft}}} w_i n_i(I) \Bigg) & \text{if } I \in \text{SM}[\Pi]; \\ 0 & \text{otherwise,} \end{cases}$$

where $n_i(I)$ is the number of ground instances of R_i that is not satisfied by I. Due to Proposition 3, under our assumption that $SM'[\Pi]$ is not empty, the probability of any interpretation I can be computed as

$$P_{\Pi}(I) = \frac{W'_{\Pi}(I)}{\sum\limits_{J \in \text{SM}[\Pi]} W'_{\Pi}(J)}.$$

In this section, we use the above equationa of weight and probability.

6.0.1 General Problem Statement

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

$$\underset{\mathbf{w}}{\text{argmax}}\ P_{\hat{\Pi}(\mathbf{w})}(O). \tag{6.1}$$

6.0.2 *Gradient Method for Learning Weights From a Complete Stable Model*

Same as in Markov Logic, there is no closed form solution for (6.1) but the gradient ascent method can be applied to find the optimal weights in an iterative manner.

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

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

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

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

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

Since the log-likelihood above is a concave function of the weights, any local maximum is a global maximum, and maximizing $P_{\Pi}(I)$ can be done by the standard gradient ascent method by updating each weight w_i by $w_i+\lambda\cdot(-n_i(I)+\underset{J\in SM[\Pi]}{E}[n_i(J)])$ until it converges.[1]

However, similar to Markov Logic, computing $\underset{J\in SM[\Pi]}{E}[n_i(J)]$ is intractable (Richardson and Domingos (2006)). In the next section, we turn to an MCMC sampling

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

method to find its approximate value.

6.0.3 Sampling Method: MC-ASP

The following is an MCMC algorithm for LP$^{\text{MLN}}$, which adapts the algorithm MC-SAT for Markov Logic Poon and Domingos (2006) by considering the penalty-based reformulation and by using an ASP solver instead of a SAT solver for sampling.

Algorithm 1 MC-ASP

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

Output: Samples $I^1, \dots, I^N$

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

When all the weights w_i of soft rules are non-positive, $1 - e^{w_i}$ (at step (b)) is in the range $[0, 1)$ and thus it validly represents a probability. At each iteration, the sample is chosen from stable models of Π, and consequently, it must satisfy all hard rules. For soft rules, the higher its weight, the less likely that it will be included in M, and thus less likely to be not satisfied by the sample generated from M.

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

Theorem 15. *The Markov chain generated by MC-ASP satisfies ergodicity and detailed balance.*[2]

Steps 1 and 2(c) of the algorithm require finding a probabilistic stable model of LP$^{\rm MLN}$, which can be computed by system LPMLN2ASP (see Section 5.1). System LPMLN2ASP turns an LP$^{\rm MLN}$ program Π into $\mathsf{lpmln2asp}^{\rm pnt}(\Pi)$ and calls ASP solver CLINGO to find the stable models of $\mathsf{lpmln2asp}^{\rm pnt}(\Pi)$, which coincide with the probabilistic stable models of Π. The weight of a stable model can be computed from the weights recorded in `unsat` atoms that are true in the stable model.

Step 2(c) also requires a uniform sampler for answer sets, which can be computed by XORRO (Gebser *et al.* (2016)).

Algorithm 2 is a weight learning algorithm for LP$^{\rm MLN}$ based on gradient ascent using MC-ASP (Algorithm 1) for collecting samples. Step 2(b) of MC-ASP requires that w_i be non-positive in order for $1 - e^{w_i}$ to represent a probability. Unlike in the Markov Logic setting, converting positive weights into non-positive weights cannot be done in LP$^{\rm MLN}$ simply by replacing $w : F$ with $-w : \neg F$, due to the difference in the FOL and the stable model semantics. Algorithm 2 converts Π into an equivalent program Π^{neg} whose rules' weights are non-positive, before calling MC-ASP. The following theorem justifies the soundness of this method.[3]

Theorem 16. *When* $\mathrm{SM}[\Pi]$ *is not empty, the program* Π^{neg} *specifies the same probability distribution as the program* Π.[4]

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

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

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

[4] Non-emptiness of $\mathrm{SM}[\Pi]$ implies that every probabilistic stable model of Π satisfies all hard rules in Π.

Algorithm 2 Algorithm for learning weights using LPMLN2ASP

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

Output: Π with learned weights.

Process:

1. Initialize the weights of soft rules $R_1, \dots, R_m$ with some initial weights $\mathbf{w}^0$.
2. Repeat the following for $j = 1, \dots$ until $max\{|w_i^j - w_i^{j-1}| : i = 1, \dots, m\} < \delta$:
   (a) Compute the stable model of $\Pi \cup O$ using LPMLN2ASP (see below); for each soft rule R_i, compute $n_i(O)$ by counting `unsat` atoms whose first argument is i (i is a rule index).
   (b) Create Π^{neg} by replacing each soft rule R_i of the form $w: \; H(\mathbf{x}) \leftarrow B(\mathbf{x})$ in Π where $w > 0$ with
   $$0: \; H(\mathbf{x}) \leftarrow B(\mathbf{x}),$$
   $$\alpha: \; \texttt{neg}(i, \mathbf{x}) \leftarrow B(\mathbf{x}), \texttt{not } H(\mathbf{x}),$$
   $$-w: \leftarrow \texttt{not neg}(i, \mathbf{x}).$$
   (c) Run MC-ASP on Π^{neg} to collect a set S of sample stable models.
   (d) For each soft rule R_i, approximate $\sum_{J \in SM[\Pi]} P_\Pi(J) n_i(J)$ with $\sum_{J \in S} n_i(J)/|S|$, where n_i is obtained from counting the number of `unsat` atoms whose first argument is i.
   (e) For each $i \in \{1, \dots, m\}$,
   $w_i^{j+1} \leftarrow w_i^j + \lambda \cdot (-n_i(O) + \sum_{J \in S} n_i(J)/|S|)$.

6.1 Extensions

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

6.1.1 Learning from Multiple Stable Models

The method described in the previous section allows only one stable model to be used as the training data. Now, suppose we have multiple stable models $I_1, \dots, I_m$ as the training data. For example, consider the parameterized program $\hat{\Pi}_{coin}$ that describes a coin, which may or may not land in the head when it is flipped,

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

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

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

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

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

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

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

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

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

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

$$I = \bigcup_{i \in \{1,\dots,m\}} \{p(\mathbf{t}, i) \mid p(\mathbf{t}) \in I_i\}. \tag{6.2}$$

The following theorem asserts that the weights of the rules in Π that are learned from the multiple stable models $I_1, \dots, I_m$ are identical to the weights of the rules in Π^m that are learned from the single stable model I that conjoins $\{I_1, \dots, I_m\}$ as in (6.2).

Theorem 17. *For any parameterized* LP$^{\text{MLN}}$ *program* $\hat{\Pi}$*, its stable models* $I_1, \dots, I_m$ *and* I *as defined as in* (6.2)*, we have*

$$\underset{\mathbf{w}}{\text{argmax}}\ P_{\hat{\Pi}^m(\mathbf{w})}(I) = \underset{\mathbf{w}}{\text{argmax}} \prod_{i \in \{1,\dots,m\}} P_{\hat{\Pi}(\mathbf{w})}(I_i).$$

Example 9. *For the program $\hat{\Pi}_{coin}$, to learn from the three stable models I_1, I_2, and I_3 defined before, we consider the program $\hat{\Pi}^3_{coin}$*

$$\alpha \ : \ \{flip(k)\}.$$
$$w \ : \ head(k) \leftarrow flip(k).$$

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

6.1.2 Learning in the Presence of Noisy Data

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

Example 10. *Consider again the program $\hat{\Pi}^m_{coin}$. Suppose one of the interpretations I_i in the training data is $\{head(i)\}$. The interpretation is not a stable model of $\hat{\Pi}^m_{coin}$.*

We obtain $\hat{\Pi}^m_{noisecoin}$ *by modifying* $\hat{\Pi}^m_{coin}$ *to allow for the noisy atom* $n(k)$ *as follows.*

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

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

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

6.1.3 Learning from Incomplete Interpretations

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

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

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

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

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

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

6.2 LP$^{\mathrm{MLN}}$ Weight Learning via Translations to Other Languages

This section considers two fragments of LP$^{\mathrm{MLN}}$, for which the parameter learning task reduces to the same tasks for Markov Logic and ProbLog.

6.2.1 *Tight* LP$^{\mathrm{MLN}}$ *Program: Reduction to MLN Weight Learning*

By Theorem 9, any tight LP$^{\mathrm{MLN}}$ program can be translated into a Markov Logic Network (MLN) by adding completion formulas Erdem and Lifschitz (2003) with the weight α. This means that the weight learning for a tight LP$^{\mathrm{MLN}}$ program can be reduced to the weight learning for an MLN.

Given a tight LP$^{\mathrm{MLN}}$ program $\Pi = \langle \mathbf{R}, \mathbf{W} \rangle$ and one (not necessarily complete) interpretation E as the training data, the MLN $Comp(\Pi)$ is obtained by adding completion formulas with weight α to Π.

The following theorem tells us that the weight assignment that maximizes the probability of the training data under LP$^{\mathrm{MLN}}$ programs is identical to the weight assignment that maximizes the probability of the same training data under an MLN $Comp(\Pi)$.

Theorem 18. *Let* L *be the Markov Logic Network* $Comp(\Pi)$ *and let* E *be a ground formula (as the training data). When* SM[Π] *is not empty,*

$$\underset{\mathbf{w}}{\text{argmax}}\ P_{\hat{\Pi}(\mathbf{w})}(E) = \underset{\mathbf{w}}{\text{argmax}}\ P_{\hat{\text{L}}(\mathbf{w})}(E).$$

($\hat{\text{L}}$ is a parameterized MLN obtained from L.*)*

Thus we may learn the weights of a tight LP$^{\text{MLN}}$ program using the existing implementations of Markov Logic, such as ALCHEMY and TUFFY.

6.2.2 Coherent LP$^{\text{MLN}}$ *Program: Reduction to Parameter Learning in ProbLog*

For another special class of LP$^{\text{MLN}}$ programs, weight learning can be reduced to weight learning in ProbLog (Fierens *et al.* (2013)).

We say an LP$^{\text{MLN}}$ program Π is *simple* if all soft rules in Π are of the form

$$w : A$$

where A is an atom, and no atoms occurring in the soft rules occur in the head of a hard rule.

We say a simple LP$^{\text{MLN}}$ program Π is k*-coherent* ($k > 0$) if, for any truth assignment to atoms that occur in Π^{soft}, there are exactly k probabilistic stable models of Π that satisfies the truth assignment. We also apply the notion of k-coherency when Π is parameterized.

Without loss of generality, we assume that no atom occurs more than once in Π^{soft}. (If one atom A occurs in multiple rules $w_1 : A, \dots, w_n : A$, these rules can be combined into $w_1 + \dots + w_n : A$.) A k-coherent LP$^{\text{MLN}}$ program Π can thus be identified with the tuple $\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle$, where $PF = (pf_1, \dots, pf_m)$ is a list of (possibly non-ground) atoms that occur as soft rules in Π, Π^{hard} is a set of hard rules in Π, and $\mathbf{w} = (w_1, \dots, w_m)$ is the list of soft rule's weights, where w_i is the weight of pf_i.

A ProbLog program can be viewed as a tuple $\langle PF, \mathbf{R}, \mathbf{pr}\rangle$ where PF is a list of atoms called *probabilistic facts*, $\mathbf{R}$ is a set of rules such that no atom that occurs in PF occurs in the head of any rule in $\mathbf{R}$, and $\mathbf{pr}$ is a list $(p_1, \dots, p_{|PF|})$, where each p_i is the probability of probabilistic atom $pf_i \in PF$. A *parameterized* ProbLog program is similarly defined, where $\mathbf{pr}$ is a list of parameters to be learned.

Given a list of probabilities $\mathbf{pr} = (p_1, \dots, p_n)$, we construct a list of weights $\mathbf{w}^{\mathbf{pr}} = (w_1, \dots, w_n)$ as follows:

$$w_i = ln(\frac{p_i}{1 - p_i}) \tag{6.3}$$

for $i \in \{1, \dots n\}$.

The following theorem asserts that weight learning on a 1-coherent LP$^{\text{MLN}}$ program can be done by weight learning on its corresponding ProbLog program.

Theorem 19. *For any 1-coherent parameterized* LP$^{\text{MLN}}$ *program* $\langle PF, P, \mathbf{w}\rangle$ *and any interpretation* T *(as the training data), we have*

$$\mathbf{w} = \underset{\mathbf{w}}{\text{argmax}}\ P_{\langle PF,P,\mathbf{w}\rangle}(T)$$

if and only if

$$\mathbf{w} = \mathbf{w}^{\mathbf{pr}} \ \textit{and}\ \mathbf{pr} = \underset{\mathbf{pr}}{\text{argmax}}\ P_{\langle PF,P,\mathbf{pr}\rangle}(T).$$

According to the theorem, to learn the weights of a 1-coherent LP$^{\text{MLN}}$ program, we can simply construct the corresponding ProbLog program, perform ProbLog weight learning, and then turn the learned probabilities into LP$^{\text{MLN}}$ weights according to (6.3).

As we will see in Chapter 7, k-coherent programs are useful for describing dynamic domains. Intuitively, each probabilistic choice leads to the same number of histories. For such a k-coherent LP$^{\text{MLN}}$ program, weight learning given a complete interpretation as the training data can be done by simply counting true and false ground instances of soft atomic facts in the given interpretation.

For an interpretation I and $c_i \in PF$, let $m_i(I)$ and $n_i(I)$ be the numbers of ground instances of c_i that is true in I and false in I, respectively.

Theorem 20. *For any k-coherent parameterized* LP$^{\text{MLN}}$ *program* $\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle$*, and any (complete) interpretation I (as the training data), we have*

$$\underset{\mathbf{w}}{\text{argmax}}\ P_{\langle PF,\Pi^{\text{hard}},\mathbf{w}\rangle}(I;\mathbf{w}) = \Big(ln\frac{m_1(I)}{n_1(I)},\dots,ln\frac{m_{|PF|}(I)}{n_{|PF|}(I)}\Big).$$

6.3 Implementation and Examples

We implemented Algorithm 2 and its extensions described above using CLINGO, LPMLN2ASP, and a near-uniform answer set sampler XORRO . The implementation LPMLN-LEARN is available at `https://github.com/ywng485/lpmln-learning` together with a manual and some examples. In this section, we show how the implementation allows for learning weights in LP$^{\text{MLN}}$ from the data enabling learning parameters in knowledge-rich domains.

For all the experiments in this section, δ is set to be 0.001. λ is fixed to 0.1 and 50 samples are generated for each call of MC-ASP. The parameters for XORRO are manually tuned to achieve the best performance for each specific example.

6.3.1 Learning Certainty Degrees of Hypotheses

The LP$^{\text{MLN}}$ weight learning algorithm can be used to learn the certainty degree of a hypothesis from the data. For example, consider a person A carrying a certain virus contacting a group of people. The virus spreads among them as people contact each other. We use the following ASP facts to specify that A carries the virus and how people contacted each other:

```
carries_virus("A").
```

```
contact("A", "B").  contact("B", "C").  ...
```

Consider two hypotheses that a person carrying the virus may cause him to have a certain disease, and the virus may spread by contact. The hypotheses can be represented in the input language of LPMLN-LEARN by the following rules, where `w(1)` and `w(2)` are parameters to be learned:

```
@w(1) has_disease(X) :- carries_virus(X).
@w(2) carries_virus(Y) :- contact(X, Y),
                          carries_virus(X).
```

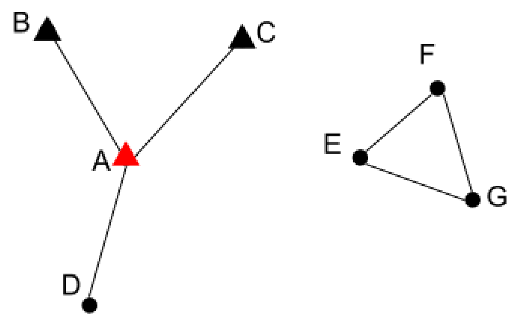
The parameterized LP$^{\text{MLN}}$ program consists of these two rules and the facts about `contact` relation. The training data specifies whether each person carries the virus and has the disease, for example:

```
:- not carries_virus("E").   :- carries_virus("H").
...
:- not has_disease("A").      :- has_disease("H").
```

The learned weights tell us how certain the data support the hypotheses. Note that the program models the transitive closure of the `carries_virus` relation, which is not properly done if the program is viewed as an MLN.[5] Learning under the MLN semantics results in weights that associate unreasonably high probabilities to people carrying virus even if they were not contacted by people with virus.

For example, consider the following graph

[5]That is, identifying the rule $H \leftarrow B$ with a formula in first-order logic $B \rightarrow H$.



where `A` is the person who initially carries the virus, triangle-shaped nodes represent people who carry virus in the evidence, and the edges denote the `contact` relation. The cluster consisting of `E`, `F`, and `G` has no contact with the cluster consisting of `A`, `B`, `C`, and `D`. The following table shows the probability of each person carrying the virus, which is derived from the weights learned in accordance with Markov Logic and LP$^{\text{MLN}}$, respectively. We use ALCHEMY for the weight learning in Markov Logic.

Person	MLN	LP$^{\text{MLN}}$	carries_virus (ground truth)
B	0.823968	0.6226904833	Y
C	0.813969	0.6226904833	Y
D	0.818968	0.6226904833	N
E	0.688981	0	N
F	0.680982	0	N
G	0.680982	0	N

As can be seen from the table, under MLN, each of `E`, `F`, `G` has a high probability of carrying the virus, which is unintuitive.

6.3.2 Learning Probabilistic Graphs from Reachability

Consider an (unstable) communication network such as the one in Figure 6.1, where each node represents a signal station that sends and receives signals. A station may fail, making it impossible for signals to go through the station. The following

LP$^{\text{MLN}}$ rules define the connectivity between two stations X and Y in session T.

```
connected(X,Y,T) :- edge(X,Y), not fail(X,T),
                                not fail(Y,T).
connected(X,Y,T) :- connected(X,Z,T), connected(Z,Y,T).
```

A specific network can be defined by specifying edge relations, such as `edge(1,2)`. Suppose we have data showing the connectivity between stations in several sessions. Based on the data, we could make decisions such as which path is most reliable to send a signal between the two stations. Under the LP$^{\text{MLN}}$ framework, this can be done by learning the weights representing the failure rate of each station. For the network in Figure 6.1, we write the following rules whose weights $\mathtt{w}(i)$ are to be learned:

```
@w(1) fail(1, T).    ...    @w(10) fail(10, T).
```

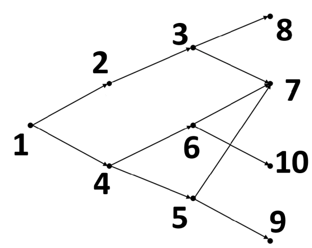


Figure 6.1: Example Communication Network

Here T is the auxiliary argument to allow learning from multiple training examples, as described in Section 6.1.1. The training example contains constraints either `:- not connected(X,Y)` for known connected stations X and Y or `:- connected(X,Y)` for known disconnected stations X and Y. Since the training data is incomplete in specifying the connectivity between the stations, we use the extension of Algorithm 2 described in Section 6.1.3. The failure rates of the stations can be obtained from the learned weights as $\frac{e^{\mathtt{w}(i)}}{e^0+e^{\mathtt{w}(i)}}$.

We execute learning on graphs with $10, 12, \ldots, 18, 20$ nodes, where the graph with

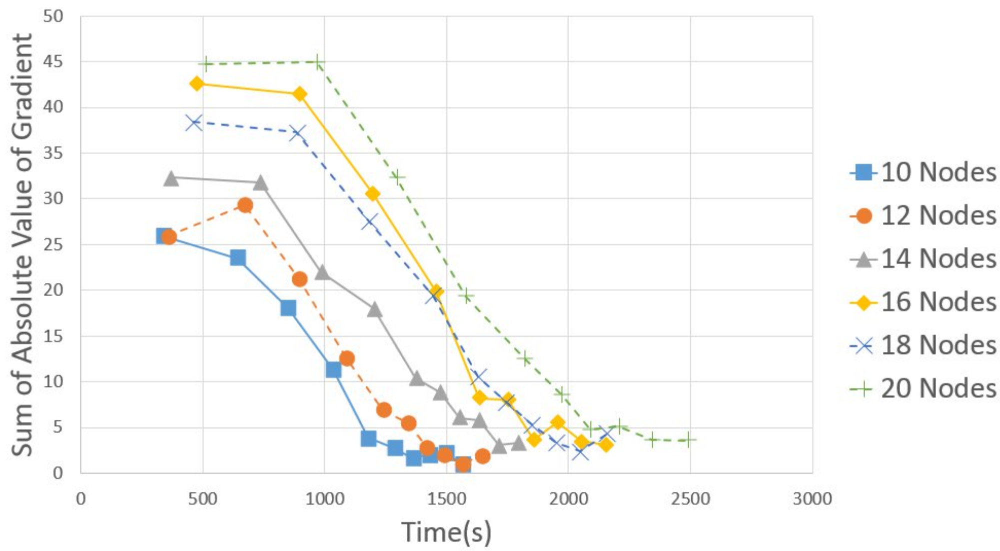


Figure 6.2: Convergence Behavior of Failure Rate Learning

10 nodes is shown in Figure 6.1. We add $1, 2, \ldots, 5$ layers of 2 nodes between Node 1 and Node $2, 4$ to obtain the other graphs, where there is an edge between every node in one layer and every node in the previous and next layer. Figure 6.2 shows the convergence behavior over time in terms of the sum of the absolute values of gradients of all weights. Running time is mostly spent by the uniform sampler for answer sets. The experiments are performed on a machine with 4 Intel(R) Core(TM) i5-2400 CPU with OS Ubuntu 14.04.5 LTS and 8 GB memory.

Figure 6.2 shows that convergence takes longer as the number of nodes increases, which is not surprising. Note that the current implementation is not very efficient. Even for graphs with $10-20$ nodes, it takes $1500-2000$ seconds to obtain a reasonable convergence. The computation bottleneck lies in the uniform sampler used in Step 2(c) of Algorithm 1 whereas creating Π_{neg} and turning LP$^{\rm MLN}$ programs into ASP programs are done instantly. The uniform sampler that we use, XORRO, follows Algorithm 2 in Gomes *et al.* (2007). It uses a fixed number of random XOR constraints to prune out a subset of stable models, and randomly select one remaining stable model to return. The process of solving for all stable models after applying XOR constraints can be very time-consuming.

In this example, it is essential that the samples are generated by an ASP solver because information about node failing needs to be correctly derived from the connectivity, which involves reasoning about the transitive closure.

As Theorem 19 indicates, this weight learning task can alternatively be done through ProbLog weight learning. We use PROBLOG,[6] an implementation of ProbLog. The performance of PROBLOG on weight learning depends on the tightness of the input program. We observed that for many tight programs, PROBLOG appears to have better scalability than our prototype LPMLN-LEARN. However, PROBLOG system does not show a consistent performance on non-tight programs, such as the encoding of the network example above, possibly due to the fact that it has to convert the input program into weighted Boolean formulas, which is expensive for non-tight programs.[7] We can identify many graph instances of the network failure example where our prototype system outperforms PROBLOG, as the density of the graph gets higher. For example, consider the graph in Figure 6.1. With the nodes fixed, as we add more edges to make the graph denser, we eventually hit a point when PROBLOG does not return a result within a reasonable time limit. Below is the statistics of several instances.

The input files to PROBLOG consist of two parts: edge lists and the part that defines the node failure rates and connectivity. The latter is different for the second column and the third column in the table. For the second column it is the same as the input to LPMLN-LEARN:

[6] `https://dtai.cs.kuleuven.be/problog/`

[7] The difference appears to be analogous to the different approaches to handling non-tight programs by answer set solvers, e.g., the translation-based approach such as ASSAT and CMODELS and the native approach such as CLINGO.

# Edges	LPMLN-LEARN	ProbLog	PROBLOG (with modified program)
10	351.237s	2.565s	0.846s
14	476.656s	2.854s	0.833s
15	740.656s	> 20 min	0.957s
20	484.348s	> 20 min	76.143s
40	304.407s	> 20 min	26.642s

```
t(_)::fail(1).     ...     t(_)::fail(10).


connected(X, Y):- edge(X, Y), not fail(X), not fail(Y).
connected(X, Y):- connected(X, Z), connected(Z, Y).
```

For the third column, we rewrite the rules to make the Boolean formula conversion easier for PROBLOG. The input program is:[8]

```
t(_)::fail(1).     ...     t(_)::fail(10).


aux(X, Y) :- edge(X, Y), not fail(X), not fail(Y).
connected(X, Y) :- aux(X, Y).
connected(X, Y) :- connected(X, Z), aux(Z, Y).
```

Although all graph instances have some cycles in the graph, the difference between the instance with 14 edges and 15 edges is the addition of one cycle. Even with the slight change in the graph, the performance of PROBLOG becomes significantly slower.

6.3.3 Learning Parameters for Abductive Reasoning about Actions

One of the successful applications of answer set programming is modeling dynamic domains. LP$^{\text{MLN}}$ can be used for extending the modeling to allow uncertainty. In Chapter 7, a high-level action language $p\mathcal{BC}+$ is defined as a shorthand notation for LP$^{\text{MLN}}$. The language allows for probabilistic diagnoses in action domains: given the

[8]This was suggested by Angelika Kimmig (personal communication)

action description and the histories where an abnormal behavior occurs, how to find the reason for the failure? There, the probabilities are specified by the user. This can be enhanced by learning the probability of the failure from the example histories using LPMLN-LEARN.[9] In this section, we show how LP$^{\text{MLN}}$ weight learning can be used for learning parameters for abductive reasoning in action domains. Due to the self-containment of the paper, instead of showing $p\mathcal{BC}+$ descriptions, we show its counterpart in LP$^{\text{MLN}}$.

Consider the robot domain described in Iwan (2002): a robot located in a building with 2 rooms `r1` and `r2` and a book that can be picked up. The robot can move to rooms, pick up the book, and put down the book. Sometimes actions may fail: the robot may fail to enter the room, may fail to pick up the book, and may drop the book when it has the book. The domain can be modeled using answer set programs, e.g., Lifschitz and Turner (1999). We illustrate how such a description can be enhanced to allow abnormalities, and how the LP$^{\text{MLN}}$ weight learning method can learn the probabilities of the abnormalities given a set of actions and their effects.

We introduce the predicate $Ab(i)$ to represent that some abnormality occurred at step i, and the predicate $Ab(AbnormalityName, i)$ to represent that a specific abnormality $AbnormalityName$ occurred at step i. The occurrences of specific abnormalities are controlled by probabilistic fact atoms and their preconditions. For example,

$$\begin{aligned} w_1 &: Pf_1(i) \\ \alpha &: Ab(EnterFailed, i) \leftarrow Pf_1(i), Ab(i). \end{aligned}$$

defines that the abnormality $EnterFailed$ occurs with probability $\frac{e^{w_1}}{e^{w_1}+1}$ (controlled

[9]ProbLog could not be used in place of LP$^{\text{MLN}}$ here because it has the requirement that every total choice leads to exactly one well founded model, and consequently does not support choice rules, which has been used in the formalization of the robot example in this section.

by the weighted atomic fact $Pf_1(i)$, which is introduced to represent the probability of the occurrence of *EnterFailed*) at time step i if there is some abnormality at time step i. Similarly we have

$$\begin{aligned} w_2 &: Pf_2(i) \\ \alpha &: Ab(DropBook, i) \leftarrow Pf_2(i), Ab(i). \\ w_3 &: Pf_3(i) \\ \alpha &: Ab(PickupFailed, i) \leftarrow Pf_3(i), Ab(i). \end{aligned}$$

When we describe the effect of actions, we need to specify "no abnormality" as part of the precondition of the effect: The location of the robot changes to room r if it goes to room r unless abnormality *EnterFailed* occurs:

$$\alpha: \; LocRobot(r, i+1) \leftarrow Goto(r, i), not \; Ab(EnterFailed, i).$$

The location of the book is the same as the location of the robot if the robot has the book:

$$\alpha \; : \; LocBook(r, i) \leftarrow LocRobot(r, i), HasBook(T, i).$$

The robot has the book if it is at the same location as the book and it picks up the book, unless abnormality *PickupFailed* occurs:

$$\begin{aligned} \alpha: \; & HasBook(\mathbf{t}, i+1) \leftarrow PickupBook(\mathbf{t}, i), \\ & \quad LocRobot(r, i), LocBook(r, i), not \; Ab(PickupFailed, i). \end{aligned}$$

The robot loses the book if it puts down the book:

$$\alpha \; : \; HasBook(\mathbf{f}, i+1) \leftarrow PutdownBook(\mathbf{t}, i).$$

The robot loses the book if abnormality *DropBook* occurs:

$$\alpha \; : \; HasBook(\mathbf{f}, i+1) \leftarrow Ab(DropBook, i).$$

The commonsense law of inertia for each fluent is specified by the following hard rules:

$$\alpha \ : \ \{LocRobot(r, i+1)\} \leftarrow LocRobot(r, i), astep(i).$$

$$\alpha \ : \ \{LocBook(r, i+1)\} \leftarrow LocBook(r, i), astep(i).$$

$$\alpha \ : \ \{HasBook(b, i+1)\} \leftarrow HasBook(b, i), astep(i).$$

For the lack of space, we skip the rules specifying the uniqueness and existence of fluents and actions, rules specifying that no two actions can occur at the same timestep, and rules specifying that the initial state and actions are exogenous.

We add the hard rule

$$\alpha \ : \ Ab(i) \leftarrow astep(i)$$

to enable abnormalities for each timestep i.

To use multiple action histories as the training data, we use the method from Section 6.1.1 and introduce an extra argument to every predicate, that represents the action history ID.

We then provide a list of 12 transitions as the training data. For example, the first transition (`ID` =1) tells us that the robot performed `goto` action to room `r2`, which failed.

```
:- not loc_robot("r1",0,1). :- not loc_book("r2",0,1).
:- not hasBook("f",0,1).    :- not goto("r2",0,1).
:- not loc_robot("r1",1,1).
```

Among the training data, `enter_failed` occurred 1 time out of 4 attempts, `pickup_failed` occurred 2 times out of 4 attempts, and `drop_book` occurred 1 time out of 4 attempts. The transitions are partially observed data in the sense that they specify only some of the fluents and actions; other facts about fluents, actions and abnormalities have to be inferred.

Note that this program is $(|A|+1)$-coherent, where $|A|$ is the number of actions (i.e., *Goto*, *PickupBook* and *DropBook*) and 1 is for no actions. We execute gradient ascent learning with 50 learning iterations and 50 sampling iterations for each learning iteration. The weights learned are

```
Rule 1:  -1.084    Rule 2:  -1.064    Rule 3:  -0.068
```

The probability of each abnormality can be computed from the weights as follows:

$$P(\texttt{enter\_failed}) = \frac{exp(-1.084)}{exp(-1.084)+1} \approx 0.253$$
$$P(\texttt{drop\_book}) = \frac{exp(-1.064)}{exp(-1.064)+1} \approx 0.257$$
$$P(\texttt{pickup\_failed}) = \frac{exp(-0.068)}{exp(-0.068)+1} \approx 0.483$$

The learned weights of `pf` atoms indicate the probability of the action failure when some abnormal situation `ab(I, ID)` happens. This allows us to perform probabilistic diagnostic reasoning in which parameters are learned from the histories of actions. For example, suppose the robot and the book were initially at `r1`. The robot executed the following actions to deliver the book from `r1` to `r2`: pick up the book; go to `r2`; put down the book. However, after the execution, it observes that the book is not at `r2`. What was the problem?

Executing system LPMLN2ASP on this encoding tells us that the most probable reason is that the robot fails at picking up the book. However, if we add that the robot itself is also not at `r2`, then LPMLN2ASP computes the most probable stable model to be the one that has the robot failed at entering `r2`.

6.4 Related Work

Parameter learning (with full or partial observability) is usually formulated as maximizing the probability of the given training evidence. For Markov logic networks, this optimization problem does not have a closed-form solution, however the log-likelihood of the training evidence is provably a concave function, and thus standard gradient ascent can be used to find the global optimum (Domingos and Lowd (2009)). On the other hand, Khot *et al.* (2015) is a method that learns weight and structure (formulas) of MLN programs at the same time. The MLNs learned are predictive models that can predict the truth value of groundings of a set of target predicates. The structure and weights of an MLN is represented as a regression tree that fits the training examples . The problem of learning structure and weights of an MLN program is thus turned into a series of relational regression problem.

For ProbLog, with full observability, the optimal probability annotation of atoms can be found by simply counting the frequency of the atoms in the training evidence. With partial observability, there is also no closed-form solution to maximizing the probability of the training evidence. In this setting, EM algorithm is used to iteratively update the parameters (Fierens *et al.* (2013)).

Sometimes the training evidence is marginal/conditional probability of certain atom, i.e., result of a query, in which case the setting is query-based learning, and the learning problem is formulated as minimizing the difference between the given probabilities of queries (target probabilities) and the probabilities of queries computed from the program. As an example, Gutmann *et al.* (2008) minimizes the mean squares difference between the actual probabilities and target probabilities.

6.5 Proofs

6.5.1 Proof of Theorem 15

Lemma 18. *For any* LP$^{\text{MLN}}$ *program* Π *and a probabilistic stable model* I *of* Π*, we have*

$$P_\Pi(I) = \frac{exp(\sum\limits_{w:R\in\Pi_I^{\text{soft}}} w)}{Z}$$

where

$$Z = \sum_{J \textit{ is a stable model of } \Pi} exp(\sum_{w:R\in\Pi_J^{\text{soft}}} w)$$

Proof. Let k be the maximum number of hard rules in Π that any interpretation can satisfy. For any interpretation J, we use $J \vDash_{SM} \Pi$ as an abbreviation of "J is a probabilistic stable model of Π".

By definition we have

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{exp(\sum\limits_{w:R\in\Pi_I} w)}{\sum\limits_{J\vDash_{SM}\Pi} exp(\sum\limits_{w:R\in\Pi_J} w)}$$

Splitting the denominator into two parts: those J's that satify k hard rules in Π and those that satisfy less hard rules, and extracting the weights of k hard rules, $k\alpha$, we have

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{exp(\sum\limits_{w:R\in\Pi_I} w)}{exp(k\alpha)\sum\limits_{\substack{J\vDash_{SM}\Pi\\ |\Pi_J^{\text{hard}}|=k}} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w) + \sum\limits_{\substack{J\vDash_{SM}\Pi\\ |\Pi_J^{\text{hard}}|<k}} exp(|\Pi_J^{\text{hard}}|\cdot\alpha) exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w)}$$

Let k' denote the number of hard rules that I satisfy. We have

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{exp(k'\alpha)exp(\sum\limits_{w:R\in\Pi_I^{\text{soft}}} w)}{exp(k\alpha)\sum\limits_{\substack{J\vDash_{SM}\Pi\\ |\Pi_J^{\text{hard}}|=k}} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w) + \sum\limits_{\substack{J\vDash_{SM}\Pi\\ |\Pi_J^{\text{hard}}|<k}} exp(|\Pi_J^{\text{hard}}|\cdot\alpha) exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w)}$$

Dividing both the numerator and the denominator by $exp(k\alpha)$, we get

$$P_{\Pi}(I) = \lim_{\alpha\to\infty} \frac{\frac{exp(k'\alpha)}{exp(k\alpha)} exp(\sum\limits_{w:R\in\Pi_I^{\text{soft}}} w)}{\sum\limits_{\substack{J\models_{SM}\Pi\\ |\Pi_J^{\text{hard}}|=k}} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w) + \frac{1}{exp(k\alpha)} \sum\limits_{\substack{J\models_{SM}\Pi\\ |\Pi_J^{\text{hard}}|<k}} exp(|\Pi_J^{\text{hard}}|\cdot\alpha) exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w)}$$

We argue that $k' = k$: Since k is the maximum number of hard rules an interpretation can satisfy, $k' \leq k$; Suppose $k' < k$. Then the above expression evaluates to 0, contradicting the fact that I is a probabilistic stable model of Π. Following the same argument, it can be seen that any stable model of Π satisfy k hard rules. So $k' = k$, and thus we have

$$\begin{aligned}
P_{\Pi}(I) &= \lim_{\alpha\to\infty} \frac{exp(\sum\limits_{w:R\in\Pi_I^{\text{soft}}} w)}{\sum\limits_{\substack{J\models_{SM}\Pi\\ |\Pi_J^{\text{hard}}|=k}} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w) + \frac{1}{exp(k\alpha)} \sum\limits_{\substack{J\models_{SM}\Pi\\ |\Pi_J^{\text{hard}}|<k}} exp(|\Pi_J^{\text{hard}}|\cdot\alpha) exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w)}\\
&= \lim_{\alpha\to\infty} \frac{exp(\sum\limits_{w:R\in\Pi_I^{\text{soft}}} w)}{\sum\limits_{\substack{J\models_{SM}\Pi\\ |\Pi_J^{\text{hard}}|=k}} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w) + \sum\limits_{\substack{J\models_{SM}\Pi\\ |\Pi_J^{\text{hard}}|<k}} \frac{exp(|\Pi_J^{\text{hard}}|\cdot\alpha)}{exp(k\alpha)} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w)}
\end{aligned}$$

For those J that satisfy less than k hard rules, $\frac{exp(|\Pi^{\text{hard}}|)}{exp(k\alpha)} \leq k-1$, so we have

$$P_{\Pi}(I) = \frac{exp(\sum\limits_{w:R\in\Pi_I^{\text{soft}}} w)}{\sum\limits_{\substack{J\models_{SM}\Pi\\ |\Pi_J^{\text{hard}}|=k}} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w)}$$

Since all stable model of Π satisfy k hard rules, we have

$$P_{\Pi}(I) = \frac{exp(\sum\limits_{w:R\in\Pi_I^{\text{soft}}} w)}{\sum\limits_{J\models_{SM}\Pi} exp(\sum\limits_{w:R\in\Pi_J^{\text{soft}}} w)}.$$

$\square$

Theorem 15 *The Markov chain generated by MC-ASP satisfies ergodicity and detailed balance.*

Proof. **Ergodicity** Firstly, for any subset M of rules generated at step 2 in Algorithm 1, the previous sample I^{j-1} is always a stable model that satisifies no rules in M, which means at least one sample can be produced at any sampling step. Secondly, it is always possible that M is an empty set. All stable models of Π are possible to be selected when M is empty set. Thus every stable model is reachable from every stable model.

Detailed Balance For any (probabilistic) stable models X and Y of Π, let $Q(X \to Y)$ denote the transition probability from X to Y (i.e., the probability that the next sample is Y given that the current sample is X), and let let $Q(X \to^M Y)$ denote the transition probability from X to Y through a particular subset of rules as the set M at step 2 in Algorithm 1. Let $Q_M(X)$ be the probability of choosing X from M. To show $P_\Pi(X)Q(X \to Y) = P_\Pi(Y)Q(Y \to X)$, we prove a stronger equation $P_\Pi(X)Q(X \to^M Y) = P_\Pi(Y)Q(Y \to^M X)$ for any $M \subseteq (\overline{\Pi}^{soft} \setminus \overline{\Pi}_X^{soft}) \cap (\overline{\Pi}^{soft} \setminus \overline{\Pi}_Y^{soft})$. By Lemma 18, we have

$$P(X) = \frac{1}{Z} \prod_{R_i \in \overline{\Pi^{\text{soft}} \setminus \Pi_X^{\text{soft}}}} e^{-w_i}$$

and

$$Q(X \to^M Y) = \prod_{R_i \in (\overline{\Pi^{\text{soft}} \setminus \Pi_X^{\text{soft}}}) \setminus M} e^{w_i} \cdot \prod_{R_i \in M} (1 - e^{w_i}) \cdot Q_M(Y).$$

Consequently we have

$$\begin{aligned}
& P(X)Q(X \to^M Y) \\
=& \frac{1}{Z} \prod_{R_i \in \overline{\Pi^{\text{soft}} \setminus \Pi_X^{\text{soft}}}} e^{-w_i} \cdot \prod_{R_i \in (\overline{\Pi^{\text{soft}} \setminus \Pi_X^{\text{soft}}}) \setminus M} e^{w_i} \cdot \prod_{R_i \in M} (1 - e^{w_i}) \cdot Q_M(Y) \\
=& \frac{1}{Z} \cdot \prod_{R_i \in M} e^{-w_i} \cdot \prod_{R_i \in M} (1 - e^{w_i}) \cdot Q_M(Y).
\end{aligned}$$

It can be seen that $Q_M(X) = Q_M(Y)$ as any stable model of Π that satisfies M is drawn with the same probability. So we have

$$
\begin{aligned}
& P(X)Q(X \to^M Y) \\
=& \frac{1}{Z} \cdot \prod_{R_i \in M} e^{-w_i} \cdot \prod_{R_i \in M} (1 - e^{w_i}) \cdot Q_M(X) \\
=& \frac{1}{Z} \prod_{R_i \in \overline{\Pi^{\text{soft}}_Y}} e^{-w_i} \cdot \prod_{R_i \in \overline{\Pi^{\text{soft}}_Y} \setminus M} e^{w_i} \cdot \prod_{R_i \in M} (1 - e^{w_i}) \cdot Q_M(Y) \\
=& P(Y)Q(Y \to^M X).
\end{aligned}
$$

□

6.5.2 Proof of Theorem 16

Lemma 19. *Assume* $\text{SM}'[\Pi]$ *is not empty. For any interpretation* I *of* Π*,*

$$\text{neg}(I) = I \cup \{neg(i, \mathbf{x}) \mid I \not\models H(\mathbf{x}) \leftarrow B(\mathbf{x}), w_i : H(\mathbf{x}) \leftarrow B(\mathbf{x}) \in \Pi\}$$

is a $1-1$ *correspondence between* $\text{SM}[\Pi]$ *and* $\text{SM}[\Pi^{neg}]$.

Proof. We divide the ground program obtained from Π^{neg} into three parts:

$$ORIGIN(\Pi) \cup NEGDEF(\Pi) \cup NEG(\Pi)$$

where

$$
\begin{aligned}
ORIGIN(\Pi) =& \{w : H(\mathbf{x}) \leftarrow B(\mathbf{x}) \mid w : H(\mathbf{x}) \leftarrow B(\mathbf{x}) \in \Pi, w \le 0\} \cup \\
& \{0 : H(\mathbf{x}) \leftarrow B(\mathbf{x}) \mid w : H(\mathbf{x}) \leftarrow B(\mathbf{x}) \in \Pi, w > 0\},
\end{aligned}
$$

$$NEGDEF(\Pi) = \{\alpha : \texttt{neg}(i, \mathbf{x}) \leftarrow B(\mathbf{x}), \texttt{not}\ H(\mathbf{x}) \mid w : H(\mathbf{x}) \leftarrow B(\mathbf{x}) \in \Pi, w > 0\}$$

and

$$NEG(\Pi) = \{-w : \leftarrow \texttt{not}\ \texttt{neg}(i, \mathbf{x}) \mid w_i : R_i \in \Pi, w > 0\}.$$

Let σ be the signature of Π, and σ_{neg} be the set

$$\{\texttt{neg}(i, \mathbf{c}) \mid w : H(\mathbf{c}) \leftarrow B(\mathbf{c}) \in Gr(\Pi), w > 0\}.$$

For any interpretation I of Π, consider $\overline{\Pi^{\text{neg}}}_{\text{neg}(I)}$. From the construction of $\text{neg}(I)$, we have

$$\overline{\Pi^{\text{neg}}}_{\text{neg}(I)} = \overline{ORIGIN(\Pi)}_I \cup \overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$$

It can be seen that

- each strongly connected component of the dependency graph of $\overline{ORIGIN(\Pi)}_I \cup \overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$ w.r.t. $\sigma \cup \sigma_{neg}$ is a subset of σ or a subset of σ_{neg};
- no atom in σ_{neg} has a strictly positive occurrence in $\overline{ORIGIN(\Pi)}_I$;
- no atom in σ has a strictly positive occurrence in $\overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$

Thus, according to the splitting theorem, $\text{neg}(I)$ is a stable model of $\overline{\Pi^{\text{neg}}}(\text{neg}(I))$ if and only if $\text{neg}(I)$ is a stable model of $\overline{ORIGIN(\Pi)}_I$ w.r.t. σ and is a stable model of $\overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$ w.r.t. σ_{neg}.

Suppose I is a probabilistic stable model of Π. We will show that $\text{neg}(I)$ is a stable model of $\overline{\Pi^{\text{neg}}}(\text{neg}(I))$.

- $\text{neg}(I)$ **is a stable model of** $\overline{ORIGIN(\Pi)}_I$ **w.r.t.** σ**.** By definition, I is a stable model of $\overline{\Pi}_I$. Since $\overline{ORIGIN(\Pi)}_I = \overline{\Pi}_I$ and I and neg(I) agrees on σ, $\text{neg}(I)$ is a stable model of $\overline{ORIGIN(\Pi)}_I$ w.r.t. σ.
- $\text{neg}(I)$ **is a stable model of** $\overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$ **w.r.t.** σ_{neg}**.** Clearly, $\text{neg}(I)$ satisfies $\overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$. From the construction of $\text{neg}(I)$, $\text{neg}(I)$ satisfies $\texttt{neg}(i, \mathbf{x})$ only if $\text{neg}(I)$ does not satisfy $H(\mathbf{x}) \leftarrow B(\mathbf{x})$. This means $\text{neg}(I)$ satisfies

$$\texttt{neg}(i, \mathbf{c}) \rightarrow B(\mathbf{c}), \texttt{not}\ H(\mathbf{c})$$

for all rules $H(\mathbf{c}) \leftarrow B(\mathbf{c})$ in Π. This is the completion of $\overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$ w.r.t. σ_{neg}. Obviously $\overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$ is tight. So $\text{neg}(I)$ is a stable model of $\overline{NEGDEF(\Pi)}_{\text{neg}(I)} \cup \overline{NEG(\Pi)}_{\text{neg}(I)}$ w.r.t. σ_{neg}.

Suppose J is a probabilistic stable model of Π^{neg}. By definition, J is a stable model of $\overline{\Pi^{\text{neg}}}(\text{neg}(I))$. By the splitting theorem, J is a stable model of $\overline{ORIGIN(\Pi)}_I$ w.r.t. σ. Let I be the interpretation of Π obtained by dropping atoms in σ_{neg} from J. Since $\overline{ORIGIN(\Pi)}_I = \overline{\Pi}_I$ and I agrees with J on σ, I is a stable model of $\overline{\Pi}_I$, and thus is a stable model of Π. $\square$

Theorem 16 *When* SM[Π] *is not empty, the program* Π^{neg} *specifies the same probability distribution as the program* Π.

Proof. We show that $P_{\Pi^{neg}}(neg(I)) = P_{\Pi}(I)$ for all interpretations I.

By Lemma 19, since $\text{neg}(I)$ defines a $1{-}1$ correspondence between the probabilistic stable models of Π and Π^{neg}, when I is not a probabilistic stable model of Π, $\text{neg}(I)$ is not a probabilistic stable model of Π^{neg}, and vice versa. So $P_{\Pi}(I) = P_{\Pi^{\text{neg}}}(\text{neg}(I)) = 0$.

For any program Π, we use $n_{\Pi,i}(I)$ to denote the number of ground instances of rule i that is satisfies by I, $m_{\Pi,i}(I)$ to denote the number of ground instances of rule i that is not satisfied by I, and $N_{\Pi,i}$ to denote the total number of ground instances of rule i.

When I is a probabilistic stable model of Π, we have

$$
\begin{aligned}
& W'_{\Pi}(I) \\
= & \ exp(\sum_{w_i:R_i\in\Pi^{\text{soft}}} w_i n_{\Pi,i}(I)) \\
= & \ \text{(Splitting rules into the ones whose weights are positive and} \\
& \quad \text{the ones whose weights are non-positive)} \\
& exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} w_i n_{\Pi,i}(I)) \cdot exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i\le 0} w_i n_{\Pi,i}(I)).
\end{aligned}
$$

$$
\begin{aligned}
& W'_{\Pi^{neg}}(\text{neg}(I)) \\
= & \ exp(\sum_{w_i:R_i\in(\Pi^{neg})^{\text{soft}}} w_i n_{\Pi^{neg},i}(\text{neg}(I))) \\
= & \ \text{(Splitting rules into the ones whose weights are positive and} \\
& \quad \text{the ones whose weights are non-positive)} \\
& exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i\le 0} w_i n_{\Pi,i}(I)) \cdot exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} -w_i m_{\Pi,i}(I)) \\
= & \ \frac{exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} w_i N_{\Pi,i}) \cdot exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i\le 0} w_i n_{\Pi,i}) \cdot exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} -w_i n_{\Pi,i}(I))}{exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} w_i N_{\Pi,i})} \\
= & \ \frac{exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} w_i N_{\Pi,i}) \cdot exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} -w_i n_{\Pi,i}(I)) \cdot exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i\le 0} w_i n_{\Pi,i})}{exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} w_i N_{\Pi,i})} \\
= & \ \frac{1}{exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} w_i N_{\Pi,i})} exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i>0} w_i n_{\Pi,i}(I)) \cdot exp(\sum_{w_i:R_i\in\Pi^{\text{soft}},w_i\le 0} w_i n_{\Pi,i}(I)) \\
\propto & \ W'_{\Pi}(I).
\end{aligned}
$$

Consequently, we have

$$P'_{\Pi}(I) = P'_{\Pi^{neg}}(\text{neg}(I)).$$

Since SM$'[\Pi]$ is not empty, by Proposition 2 in Lee and Wang (2016), we have

$$P_{\Pi}(I) = P_{\Pi^{neg}}(\text{neg}(I)).$$

□

6.5.3 Proof of Theorem 17

Theorem 17 *For any parameterized* $\mathrm{LP}^{\mathrm{MLN}}$ *program* $\hat{\Pi}$, *its stable models* $I_1, \dots, I_m$ *and* I *as defined as in* (6.2), *we have*

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

Proof. For any weight vector $\mathbf{w}$, we show

$$P_{\hat{\Pi}^m(\mathbf{w})}(D) = \prod_{i\in\{1,\dots,m\}} P_{\hat{\Pi}(\mathbf{w})}(D_i)$$

by induction. For any integer $1 \leq u \leq m$, we use D^u to denote the interpretation

$$D^u = \{p(\mathbf{x}, j) \mid p(\mathbf{x}, j) \in D, j \leq u\}$$

Note that $D^m = D$.

Base Case: Suppose $m = 1$. It is trivial that we have

$$P_{\hat{\Pi}^1(\mathbf{w})}(D^1) = \prod_{i\in\{1\}} P_{\hat{\Pi}(\mathbf{w})}(D_i)$$

For $m > 1$, as I.H., we assume

$$P_{\hat{\Pi}^{m-1}(\mathbf{w})}(D^{m-1}) = \prod_{i\in\{1,\dots,m-1\}} P_{\hat{\Pi}(\mathbf{w})}(D_i)$$

We divide $\hat{\Pi}^m(\mathbf{w})$ into two disjoint subsets:

$$\hat{\Pi}^m(\mathbf{w}) = \hat{\Pi}^{m-1}(\mathbf{w}) \cup \hat{\Pi}(\mathbf{w})[x = m]$$

where $\hat{\Pi}[x = m](\mathbf{w})$ is the program obtained from $\hat{\Pi}(\mathbf{w})$ by appending one more argument whose value is m to the list of argument of every occurrence of every predicate in $\hat{\Pi}(\mathbf{w})$. Clearly, the intersection between the set of atoms that occur in

$gr(\hat{\Pi}^{m-1}(\mathbf{w}))$ and that occur $gr(\hat{\Pi}(\mathbf{w})[x=m])$ is empty. According to Definition 12 in Wang *et al.* (2018), $gr(\hat{\Pi}^{m}(\mathbf{w}))$ is independently divisible and $gr(\hat{\Pi}^{m-1}(\mathbf{w}))$ and $gr(\hat{\Pi}(\mathbf{w})[x=m])$ are independent programs w.r.t. $gr(\hat{\Pi}^{m}(\mathbf{w}))$.

By Corollary 3 in Wang *et al.* (2018), we have

$$\begin{aligned} P_{\hat{\Pi}^m(\mathbf{w})}(D^m) &= P_{\hat{\Pi}^{m-1}(\mathbf{w})}(D^{m-1}) \cdot P_{\hat{\Pi}(\mathbf{w})[x=m]}(D^m \setminus D^{m-1}) \\ &= P_{\hat{\Pi}^{m-1}(\mathbf{w})}(D^{m-1}) \cdot P_{\hat{\Pi}(\mathbf{w})[x=m]}(D_m) \end{aligned}$$

By I.H., we have

$$\begin{aligned} P_{\hat{\Pi}^m(\mathbf{w})}(D^m) &= \prod_{i\in\{1,\dots,m-1\}} P_{\hat{\Pi}(\mathbf{w})}(D_i) \cdot P_{\hat{\Pi}(\mathbf{w})[x=m]}(D_m) \\ &= \prod_{i\in\{1,\dots,m\}} P_{\hat{\Pi}(\mathbf{w})}(D_i). \end{aligned}$$

□

6.5.4 Proof of Theorem 18

Theorem 18 *Let* L *be the Markov Logic Network* $Comp(\Pi)$ *and let* E *be a ground formula (as the training data). When* SM[Π] *is not empty,*

$$\underset{\mathbf{w}}{\text{argmax}}\ P_{\hat{\Pi}(\mathbf{w})}(E) = \underset{\mathbf{w}}{\text{argmax}}\ P_{\hat{\mathrm{L}}(\mathbf{w})}(E).$$

($\hat{\mathrm{L}}$ *is a parameterized Markov Logic Network obtained from* L*.)*

Proof. Easily follows from Theorem 8. □

6.5.5 Proof of Theorem 19 and Theorem 20

Lemma 20. *For any 1-coherent* LP$^{\text{MLN}}$ *program* $\langle PF, P, \mathbf{w}\rangle$*, we have*

$$P_{\langle PF,P,\mathbf{w}\rangle}(I) = P_{\langle PF,P,\mathbf{pr}\rangle}(I)$$

for any interpretation I *and* $\mathbf{w} = \mathbf{w}^{\mathbf{pr}}$

Proof. Similar to the proof of Theorem 10. $\square$

Theorem 19 *For any 1-coherent parameterized* LP^{MLN} *program* $\langle PF, P, \mathbf{w}\rangle$ *and any interpretation* T *(as the training data), we have*

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}}\ P_{\langle PF,P,\mathbf{w}\rangle}(T)$$

if and only if

$$\mathbf{w} = \mathbf{w}^{\mathbf{pr}} \text{ and } \mathbf{pr} = \underset{\mathbf{pr}}{\operatorname{argmax}}\ P_{\langle PF,P,\mathbf{pr}\rangle}(T).$$

Proof. Easily follows from Lemma 20. $\square$

Proposition 8. *For any k-coherent* LP^{MLN} *program* $\Pi = \langle PF, \Pi^{hard}, \mathbf{w}\rangle$ *and any interpretation* I*, we have*

$$P_{\Pi}(I) = \frac{1}{k \cdot \prod\limits_{pf_j \in PF} (1 + e^{w_j})} W_{\Pi}(I).$$

Proof. We show that the normalization factor is constant $k \cdot \prod\limits_{pf_j \in PF} (1 + e^{w_j})$, i.e.,

$$\sum_{I \text{ is an interpretation of } \Pi} W_{\Pi}(I) = k \cdot \prod_{pf_j \in PF} (1 + e^{w_j}).$$

Let $pf_1, \dots, pf_m \in PF$ be the soft atoms. Let TC_{Π} be the set of all truth assignments to atoms in PF.

$$
\begin{aligned}
&\sum_{I \text{ is an interpretation of } \Pi} W_\Pi(I)\\
=&\sum_{I \in SM[\Pi]} W_\Pi(I)\\
=&\sum_{tc \in TC_\Pi} k \cdot \prod_{tc \models pf_i} exp(w_i) \cdot \prod_{tc \not\models pf_j} exp(0)\\
=&k \sum_{tc \in TC_\Pi} \cdot \prod_{tc \models pf_i} exp(w_i) \cdot \prod_{tc \not\models pf_j} exp(0)\\
=&k \cdot (e^{w_1} \prod_{\substack{tc \in TC_\Pi\\ tc \models pf_i\\ i \neq 1}} e^{w_i} \cdot \prod_{\substack{tc \in TC_\Pi\\ tc \not\models pf_i\\ i \neq 1}} e^0 + e^0 \prod_{\substack{tc \in TC_\Pi\\ tc \models pf_i\\ i \neq 1}} e^{w_i} \cdot \prod_{\substack{tc \in TC_\Pi\\ tc \not\models pf_i\\ i \neq 1}} e^0)\\
=&k \cdot (e^{w_2} \cdot (e^{w_1} \prod_{\substack{tc \in TC_\Pi\\ tc \models pf_i\\ i \neq 1\\ i \neq 2}} e^{w_i} \cdot \prod_{\substack{tc \in TC_\Pi\\ tc \not\models pf_i\\ i \neq 1\\ i \neq 2}} e^0 + e^0 \prod_{\substack{tc \in TC_\Pi\\ tc \models pf_i\\ i \neq 1\\ i \neq 2}} e^{w_i} \cdot \prod_{\substack{tc \in TC_\Pi\\ tc \not\models pf_i\\ i \neq 1\\ i \neq 2}} e^0)+\\
&\quad e^0 \cdot (e^{w_1} \prod_{\substack{tc \in TC_\Pi\\ tc \models pf_i\\ i \neq 1\\ i \neq 2}} e^{w_i} \cdot \prod_{\substack{tc \in TC_\Pi\\ tc \not\models pf_i\\ i \neq 1\\ i \neq 2}} e^0 + e^0 \prod_{\substack{tc \in TC_\Pi\\ tc \models pf_i\\ i \neq 1\\ i \neq 2}} e^{w_i} \cdot \prod_{\substack{tc \in TC_\Pi\\ tc \not\models pf_i\\ i \neq 1\\ i \neq 2}} e^0))\\
=&\dots\\
=&k \cdot (\sum_{p_1 \in \{e^{w_1}, 1\}} p_1 \dots \sum_{p_2 \in \{e^{w_2}, 1\}} p_2 \sum_{p_{m-1} \in \{e^{w_{m-1}}, 1\}} p_{m-1} \cdot (e^{w_m} + 1))\\
=&k \cdot (e^{w_m} + 1)(\sum_{p_1 \in \{e^{w_1}, 1\}} p_1 \dots \sum_{p_2 \in \{e^{w_2}, 1\}} p_2 \sum_{p_{m-1} \in \{e^{w_{m-1}}, 1\}} p_{m-1})\\
=&k \cdot (e^{w_m} + 1)(e^{w_{m-1}} + 1)(\sum_{p_1 \in \{e^{w_1}, 1\}} p_1 \dots \sum_{p_2 \in \{e^{w_2}, 1\}} p_2 \sum_{p_{m-2} \in \{e^{w_{m-2}}, 1\}} p_{m-1})\\
=&\dots\\
=&k \cdot \prod_{pf_j \in PF} (1 + e^{w_j}).
\end{aligned}
$$

□

Proposition 9. *For any k-coherent* LP$^{\text{MLN}}$ *program* $\mathbf{\Pi} = \langle PF, \Pi^{hard}, \mathbf{w} \rangle$ *and any* *interpretation I, we have*

$$Pr_{\mathbf{\Pi}}(I) = \begin{cases} \frac{1}{k}\prod\limits_{c_i \in PF} Pr_{\mathbf{\Pi}}(c)^{m_i(I)} \cdot (1 - Pr_{\mathbf{\Pi}}(c_i))^{n_i(I)} & \\ & \textit{if } I \textit{ is a stable model of } \mathbf{\Pi} \\ 0 & \textit{otherwise} \end{cases}$$

Proof. Easily proven from Proposition 12. □

Proposition 10. *For any k-coherent* LP$^{\text{MLN}}$ *program $\mathbf{\Pi} = \langle PF, \Pi^{hard}, \mathbf{w}\rangle$, we have*

$$Pr_{\mathbf{\Pi}}(pf_i) = \frac{exp(w_i)}{exp(w_i) + 1}$$

for any $pf_i \in PF$ and the corresponding weight w_i.

Proof. By Proposition 12 we have

$$\begin{aligned}
& Pr_{\mathbf{\Pi}}(pf_i) \\
= & \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models pf_i}} \frac{\prod\limits_{I \models pf_j, pf_j \in PF} e^{w_j} \cdot \prod\limits_{I \not\models pf_j, pf_j \in PF} e^0}{k \cdot \prod\limits_{pf_j \in PF} (1 + e^{w_j})} \\
= & \frac{e^{w_i}}{e^{w_i} + 1} \cdot \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models pf_i}} \frac{\prod\limits_{I \models pf_j, pf_j \in PF, j \neq i} e^{w_j} \cdot \prod\limits_{I \not\models pf_j, pf_j \in PF} e^0}{k \cdot \prod\limits_{\substack{pf_j \in PF \\ j \neq i}} (1 + e^{w_j})} \\
= & \frac{e^{w_i}}{e^{w_i} + 1} \cdot k \sum_{I \text{ is a truth assignment to } PF \setminus \{pf_i\}} \frac{\prod\limits_{I \models pf_j, pf_j \in PF, j \neq i} e^{w_j} \cdot \prod\limits_{I \not\models pf_j, pf_j \in PF} e^0}{k \cdot \prod\limits_{\substack{pf_j \in PF \\ j \neq i}} (1 + e^{w_j})} \\
= & \frac{e^{w_i}}{e^{w_i} + 1} \cdot \frac{\sum\limits_{I \text{ is a truth assignment to } PF \setminus \{pf_i\}} (\prod\limits_{I \models pf_j, pf_j \in PF, j \neq i} e^{w_j} \cdot \prod\limits_{I \not\models pf_j, pf_j \in PF} e^0)}{\prod\limits_{\substack{pf_j \in PF \\ j \neq i}} (1 + e^{w_j})} \\
= & \frac{e^{w_i}}{e^{w_i} + 1} \cdot 1 \\
= & \frac{e^{w_i}}{e^{w_i} + 1}.
\end{aligned}$$

□

Theorem 20 *For any k-coherent parameterized* LP^{MLN} *program* $\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle$, *and an interpretation T as the training data, we have*

$$\underset{\mathbf{w}}{\text{argmax}}\ P_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w}) = (ln\frac{m_1(T)}{n_1(T)}, \dots, ln\frac{m_{|PF|}(T)}{n_{|PF|}(T)}).$$

Proof. We have

$$\begin{aligned}
&P_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w}) \\
=&(\text{Proposition 9}) \\
&\frac{1}{k}\prod_{c_i \in PF} Pr_{\mathbf{\Pi}}(c)^{m_i(I)} \cdot (1 - Pr_{\mathbf{\Pi}}(c_i))^{n_i(I)} \\
=&(\text{Proposition 14}) \\
&\frac{1}{k}\prod_{c_i \in PF} (\frac{exp(w_i)}{exp(w_i)+1})^{m_i(I)} \cdot (1 - \frac{exp(w_i)}{exp(w_i)+1})^{n_i(I)}
\end{aligned}$$

$$\begin{aligned}
&lnP_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w}) \\
=&ln\frac{1}{k} + \sum_{c_i \in PF} m_i(I)(w_i - ln(exp(w_i)+1)) + \\
&n_i(I)(ln1 - ln(exp(w_i+1))
\end{aligned}$$

Since $P_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w})$ is concave w.r.t. $w_i \in \mathbf{w}$, the value of w_i that maximizes $P_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w})$ can be obtained by solving

$$\frac{\partial lnP_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w})}{\partial w_i} = 0.$$

For any $w_j \in \mathbf{w}$, we have

$$\begin{aligned}
&\frac{\partial lnP_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w})}{\partial w_j} \\
=&m_j(I)(1 - \frac{exp(w_j)}{exp(w_j)+1}) - n_j(I)\frac{exp(w_j)}{exp(w_j)+1}
\end{aligned}$$

$$\frac{\partial ln P_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w})}{\partial w_i} = 0$$

is equivalent to

$$m_j(I)(1 - \frac{exp(w_j)}{exp(w_j) + 1}) = n_j(I)\frac{exp(w_j)}{exp(w_j) + 1}$$

$$\Longleftrightarrow$$

$$\frac{e^{w_j}}{e^{w_j} + 1} = \frac{m_j}{m_j + n_j}$$

$$\Longleftrightarrow$$

$$w_j = ln\frac{m_j}{n_j}$$

So we have

$$argmax P_{\langle PF, \Pi^{\text{hard}}, \mathbf{w}\rangle}(T; \mathbf{w}) = (ln\frac{m_1}{n_1}, \ldots, ln\frac{m_{|PF|}}{n_{|PF|}}).$$

□

Chapter 7

PROBABILISTIC ACTION LANGUAGE $p\mathcal{BC}+$

One of the successful applications of ASP is in conveniently representing transition systems and reasoning about paths in them. However, such a representation does not distinguish which path is more probable than others. By augmenting the known ASP representations of transition systems with weights, LP$^{\text{MLN}}$ semantics gives an intuitive encoding of probabilistic transition systems. Just like action languages such as $\mathcal{BC}+$ can be defined in terms of translation to ASP programs, in this section, we show that a probabilistic extension of $\mathcal{BC}+$, called $p\mathcal{BC}+$ can be defined in terms of translation to LP$^{\text{MLN}}$ programs. We will also illustrate that probabilistic reasoning about transition systems, such as prediction and postdiction, as well as probabilistic diagnosis for dynamic domains, can be modeled in $p\mathcal{BC}+$ and computed by LP$^{\text{MLN}}$ solvers such as LPMLN2ASP and LPMLN2MLN.

7.1 Syntax of $p\mathcal{BC}+$

We assume a propositional signature σ as defined in Section 3.3. We further assume that the signature of an action description is divided into four groups: *fluent constants*, *action constants*, *pf (probability fact) constants* and *initpf (initial probability fact) constants*. Fluent constants are further divided into *regular* and *statically determined*. The domain of every action constant is Boolean. A *fluent formula* is a formula such that all constants occurring in it are fluent constants.

The following definition of $p\mathcal{BC}+$ is based on the definition of $\mathcal{BC}+$ language from Babb and Lee (2015).

A *static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{7.1}$$

where F and G are fluent formulas.

A *fluent dynamic law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{7.2}$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined fluent constants and H does not contain initpf constants.

A *pf constant declaration* is an expression of the form

$$\textbf{caused } c = \{v_1 : p_1, \ldots, v_n : p_n\} \tag{7.3}$$

where c is a pf constant with domain $\{v_1, \ldots, v_n\}$, $0 < p_i < 1$ for each $i \in \{1, \ldots, n\}$[1], and $p_1 + \cdots + p_n = 1$. In other words, (7.3) describes the probability distribution of c.

An *initpf constant declaration* is an expression of the form (7.3) where c is an initpf constant.

An *initial static law* is an expression of the form

$$\textbf{initially } F \textbf{ if } G \tag{7.4}$$

where F is a fluent constant and G is a formula that contains neither action constants nor pf constants.

[1]We require $0 < p_i < 1$ for each $i \in \{1, \ldots, n\}$ for the sake of simplicity. On the other hand, if $p_i = 0$ or $p_i = 1$ for some i, that means either v_i can be removed from the domain of c or there is not really a need to introduce c as a pf constant. So this assumption does not really sacrifice expressivity.

A *causal law* is a static law, a fluent dynamic law, a pf constant declaration, an initpf constant declaration, or an initial static law. An *action description* is a finite set of causal laws.

We use σ^{fl} to denote the set of fluent constants, σ^{act} to denote the set of action constants, σ^{pf} to denote the set of pf constants, and σ^{initpf} to denote the set of initpf constants. For any signature σ' and any $i \in \{0, \dots, m\}$, we use $i : \sigma'$ to denote the set $\{i : a \mid a \in \sigma'\}$.

By $i : F$ we denote the result of inserting $i :$ in front of every occurrence of every constant in formula F. This notation is straightforwardly extended when F is a set of formulas.

Example 11. *The following is an action description in p$\mathcal{BC}$+ for the transition system shown in Figure 7.1, P is a Boolean regular fluent constant, and A is an action constant. Action A toggles the value of P with probability* 0.8*. Initially, P is true with probability* 0.6 *and false with probability* 0.4*. We call this action description PSD. (x is a schematic variable that ranges over $\{\mathbf{t}, \mathbf{f}\}$.)*

caused P **if** $\top$ **after** $\sim P \wedge A \wedge Pf$,

caused $\sim P$ **if** $\top$ **after** $P \wedge A \wedge Pf$,

caused $\{P\}^{\text{ch}}$ **if** $\top$ **after** P,

caused $\{\sim P\}^{\text{ch}}$ **if** $\top$ **after** $\sim P$,

caused $Pf = \{\mathbf{t} : 0.8, \mathbf{f} : 0.2\}$,

caused $InitP = \{\mathbf{t} : 0.6, \mathbf{f} : 0.4\}$,

initially $P = x$ **if** $InitP = x$.

7.2 Semantics of p$\mathcal{BC}$+

Given a non-negative integer m denoting the maximum length of histories, the semantics of an action description D in p$\mathcal{BC}$+ is defined by a reduction to multi-valued probabilistic program $Tr(D, m)$, which is the union of two subprograms D_m and D_{init} as defined below.

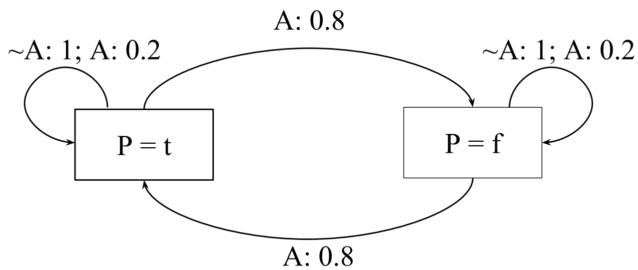


Figure 7.1: A Transition System with Probabilistic Transitions

For an action description D of a signature σ, we define a sequence of multi-valued probabilistic program $D_0, D_1, \ldots$, so that the stable models of D_m can be identified with the paths in the transition system described by D. The signature σ_m of D_m consists of atoms of the form $i : c = v$ such that

- for each fluent constant c of D, $i \in \{0, \ldots, m\}$ and $v \in Dom(c)$,
- for each action constant or pf constant c of D, $i \in \{0, \ldots, m-1\}$ and $v \in Dom(c)$.

For $x \in \{act, fl, pf\}$, we use σ_m^x to denote the subset of σ_m

$$\{i : c = v \mid i : c = v \in \sigma_m \text{ and } c \in \sigma^x\}.$$

For $i \in \{0, \ldots, m\}$, we use $i : \sigma^x$ to denote the subset of σ_m^x

$$\{i : c = v \mid i : c = v \in \sigma_m^x\}.$$

We define D_m to be the multi-valued probabilistic program $\langle PF, \Pi \rangle$, where Π is the conjunction of

$$i : F \leftarrow i : G \tag{7.5}$$

for every static law (7.1) in D and every $i \in \{0, \ldots, m\}$,

$$i{+}1 : F \leftarrow (i{+}1 : G) \wedge (i : H) \tag{7.6}$$

for every fluent dynamic law (7.2) in D and every $i \in \{0, \dots, m-1\}$,

$$\{0\!:\!c = v\}^{\text{ch}} \tag{7.7}$$

for every regular fluent constant c and every $v \in Dom(c)$,

$$\{i : c = \mathbf{t}\}^{\text{ch}}, \quad \{i : c = \mathbf{f}\}^{\text{ch}} \tag{7.8}$$

for every action constant c, and PF consists of

$$p_1 ::\ \ i : pf = v_1 \mid \cdots \mid p_n ::\ \ i : pf = v_n \tag{7.9}$$

($i = 0, \dots, m-1$) for each pf constant declaration (7.3) in D that describes the probability distribution of pf.

In addition, we define the program D_{init}, whose signature is $0 : \sigma^{initpf} \cup 0 : \sigma^{fl}$. D_{init} is the multi-valued probabilistic program

$$D_{init} = \langle PF^{init}, \Pi^{init} \rangle$$

where Π^{init} consists of the rule

$$\bot \leftarrow \neg(0\!:\!F) \wedge 0\!:\!G$$

for each initial static law (7.4), and PF^{init} consists of

$$p_1 ::\ \ 0\!:\!pf = v_1 \quad | \quad \dots \quad | \quad p_n ::\ \ 0\!:\!pf = v_n$$

for each initpf constant declaration (7.3).

We define $Tr(D, m)$ to be the union of the two multi-valued probabilistic program $\langle PF \cup PF^{init}, \Pi \cup \Pi^{init} \rangle$.

Example 12. *For the action description PSD in Example 11, PSD_{init} is the following multi-valued probabilistic program ($x \in \{\mathbf{t}, \mathbf{f}\}$):*

$$0.6 ::\ \ 0\!:\!InitP \ \mid\ 0.4 ::\ \ 0\!:\!\sim\! InitP$$

$$\bot \leftarrow \neg(0\!:\!P\!=\!x) \wedge 0 : InitP\!=\!x.$$

and PSD_m is the following multi-valued probabilistic program (i is a schematic variable that ranges over $\{1, \dots, m-1\}$):

$$
\begin{array}{ll}
0.8 :: \ i : Pf \ \mid \ 0.2 :: \ i :\sim Pf & \{i : A\}^{\rm ch} \\
i{+}1 : P \leftarrow i :\sim P \wedge i : A \wedge i : Pf & \{i :\sim A\}^{\rm ch} \\
i{+}1 :\sim P \leftarrow i : P \wedge i : A \wedge i : Pf & \{0 : P\}^{\rm ch} \\
\{i{+}1 : P\}^{\rm ch} \leftarrow i : P & \{0 :\sim P\}^{\rm ch} \\
\{i{+}1 :\sim P\}^{\rm ch} \leftarrow i :\sim P &
\end{array}
$$

For any $\mathrm{LP}^{\mathrm{MLN}}$ program Π of signature σ and a value assignment I to a subset σ' of σ, we say I is a *residual (probabilistic) stable model* of Π if there exists a value assignment J to $\sigma \setminus \sigma'$ such that $I \cup J$ is a (probabilistic) stable model of Π.

For any value assignment I to constants in σ, by $i\!:\!I$ we denote the value assignment to constants in $i\!:\!\sigma$ so that $i\!:\!I \models (i\!:\!c) = v$ iff $I \models c = v$.

We define a *state* as an interpretation I^{fl} of σ^{fl} such that $0 : I^{fl}$ is a residual (probabilistic) stable model of D_0. A *transition* of D is a triple $\langle s, e, s' \rangle$ where s and s' are interpretations of σ^{fl} and e is a an interpretation of σ^{act} such that $0\!:\!s \cup 0\!:\!e \cup 1 : s'$ is a residual stable model of D_1. A *pf-transition* of D is a pair $(\langle s, e, s' \rangle, pf)$, where pf is a value assignment to σ^{pf} such that $0\!:\!s \cup 0\!:\!e \cup 1 : s' \cup 0\!:\!pf$ is a stable model of D_1.

A *probabilistic transition system* $T(D)$ represented by a probabilistic action description D is a labeled directed graph such that the vertices are the states of D, and the edges are obtained from the transitions of D: for every transition $\langle s, e, s' \rangle$ of D, an edge labeled $e : p$ goes from s to s', where $p = Pr_{D_m}(1\!:\!s' \mid 0\!:\!s, 0\!:\!e)$. The number p is called the *transition probability* of $\langle s, e, s' \rangle$.

The soundness of the definition of a probabilistic transition system relies on the following proposition.

Proposition 11. *For any transition $\langle s, e, s' \rangle$, s and s' are states.*

We make the following simplifying assumptions on action descriptions:

1. **No Concurrency**: For all transitions $\langle s, e, s' \rangle$, we have $e(a) = t$ for at most one $a \in \sigma^{act}$;

2. **Nondeterministic Transitions are Controlled by pf constants**: For any state s, any value assignment e of σ^{act} such that at most one action is true, and any value assignment pf of σ^{pf}, there exists exactly one state s' such that $(\langle s, e, s' \rangle, pf)$ is a pf-transition;

3. **Nondeterminism on Initial States are Controlled by Initpf constants**: Given any assignment pf_{init} of σ^{initpf}, there exists exactly one assignment fl of σ^{fl} such that $0\!:\!pf_{init} \cup 0\!:\!fl$ is a stable model of $D_{init} \cup D_0$.

For any state s, any value assignment e of σ^{act} such that at most one action is true, and any value assignment pf of σ^{pf}, we use $\phi(s, e, pf)$ to denote the state s' such that $(\langle s, a, s' \rangle, pf)$ is a pf-transition (According to Assumption 2, such s' must be unique). For any interpretation I, $i \in \{0, \dots, m\}$ and any subset σ' of σ, we use $I|_{i:\sigma'}$ to denote the value assignment of I to atoms in $i : \sigma'$. Given any value assignment TC of $0\!:\!\sigma^{initpf} \cup \sigma_m^{pf}$ and a value assignment A of σ_m^{act}, we construct an interpretation $I_{TC \cup A}$ of $Tr(D, m)$ that satisfies $TC \cup A$ as follows:

- For all atoms p in $\sigma_m^{pf} \cup 0\!:\!\sigma^{initpf}$, we have $I_{TC \cup A}(p) = TC(p)$;

- For all atoms p in σ_m^{act}, we have $I_{TC \cup A}(p) = A(p)$;

- $(I_{TC \cup A})|_{0:\sigma^{fl}}$ is the assignment such that $(I_{TC \cup A})|_{0:\sigma^{fl} \cup 0:\sigma^{initpf}}$ is a stable model of $D_{init} \cup D_0$.

- For each $i \in \{1, \dots, m\}$,

$$(I_{TC\cup A})|_{i:\sigma^{fl}} = \phi((I_{TC\cup A})|_{(i-1):\sigma^{fl}}, (I_{TC\cup A})|_{(i-1):\sigma^{act}}, (I_{TC\cup A})|_{(i-1):\sigma^{pf}}).$$

By Assumptions 2 and 3, the above construction produces a unique interpretation.

It can be seen that in the multi-valued probabilistic program $Tr(D, m)$ translated from D, the probabilistic constants are $0 : \sigma^{initpf} \cup \sigma^{pf}_m$. We thus call the value assignment of an interpretation I on $0 : \sigma^{initpf} \cup \sigma^{pf}_m$ the *total choice* of I. The following theorem asserts that the probability of a stable model under $Tr(D, m)$ can be computed by simply dividing the probability of the total choice associated with the stable model by the number of choice of actions.

Theorem 21. *For any value assignment TC of $0 : \sigma^{initpf} \cup \sigma^{pf}_m$ and any value assignment A of σ^{act}_m, there exists exactly one stable model $I_{TC\cup A}$ of $Tr(D, m)$ that satisfies $TC \cup A$, and the probability of $I_{TC\cup A}$ is*

$$Pr_{Tr(D,m)}(I_{TC\cup A}) = \frac{\prod\limits_{c=v\in TC} M(c = v)}{(|\sigma^{act}| + 1)^m}.$$

The following theorem tells us that the conditional probability of transiting from a state s to another state s' with action e remains the same for all timesteps, i.e., the conditional probability of $i+1 : s'$ given $i : s$ and $i : e$ correctly represents the transition probability from s to s' via e in the transition system.

Theorem 22. *For any state s and s', and action e, we have*

$$Pr_{Tr(D,m)}(i+1:s' \mid i : s, i : e) = Pr_{Tr(D,m)}(j+1:s' \mid j : s, j : e)$$

for any $i, j \in \{0, \dots, m-1\}$ such that $Pr_{Tr(D,m)}(i : s) > 0$ and $Pr_{Tr(D,m)}(j : s) > 0$.

For every subset X_m of $\sigma_m \setminus \sigma^{pf}_m$, let $X^i (i < m)$ be the triple consisting of

- the set consisting of atoms A such that $i : A$ belongs to X_m and $A \in \sigma^{fl}$;

- the set consisting of atoms A such that $i : A$ belongs to X_m and $A \in \sigma^{act}$;
- the set consisting of atoms A such that $i{+}1{:}A$ belongs to X_m and $A \in \sigma^{fl}$.

Let $p(X^i)$ be the transition probability of X^i, s_0 is the interpretation of σ_0^{fl} defined by X^0, and e_i be the interpretations of $i : \sigma^{act}$ defined by X^i.

Since the transition probability remains the same, the probability of a path given a sequence of actions can be computed from the probabilities of transitions.

Corollary 2. *For every $m \geq 1$, X_m is a residual (probabilistic) stable model of $Tr(D,m)$ iff $X^0, \ldots, X^{m-1}$ are transitions of D and $0{:}s_0$ is a residual stable model of D_{init}. Furthermore,*

$$Pr_{Tr(D,m)}(X_m \mid 0{:}e_0, \ldots, m-1{:}e_{m-1}) = p(X^0) \times \cdots \times p(X^m) \times Pr_{Tr(D,m)}(0{:}s_0).$$

Example 13. *Consider the simple transition system with probabilistic effects in Example 11. Suppose a is executed twice. What is the probability that P remains true the whole time? With Corollary 2 this can be computed as follows:*

$$\begin{aligned}
&Pr(2 : P = \mathbf{t}, 1{:}P = \mathbf{t}, 0{:}P = \mathbf{t} \mid 0{:}A = \mathbf{t}, 1{:}A = \mathbf{t}) \\
&= p(\langle P = \mathbf{t}, A = \mathbf{t}, P = \mathbf{t}\rangle) \cdot p(\langle P = \mathbf{t}, A = \mathbf{t}, P = \mathbf{t}\rangle) \cdot Pr_{Tr(D,m)}(0{:}P = \mathbf{t}) \\
&= \ 0.2 \times 0.2 \times 0.6 = 0.024.
\end{aligned}$$

7.3 $p\mathcal{BC}+$ Action Descriptions and Probabilistic Reasoning

In this section, we illustrate how the probabilistic extension of the reasoning tasks discussed in Giunchiglia *et al.* (2004), i.e., prediction, postdiction and planning, can be represented in $p\mathcal{BC}+$ and automatically computed using LPMLN2ASP (see Section 5.1). Consider the following probabilistic variation of the well-known Yale Shooting Problem: There are two (deaf) turkeys: a fat turkey and a slim turkey. Shooting at a turkey may fail to kill the turkey. Normally, shooting at the slim turkey has 0.6

chance to kill it, and shooting at the fat turkey has 0.9 chance. However, when a turkey is dead, the other turkey becomes alert, which decreases the success probability of shooting. For the slim turkey, the probability drops to 0.3, whereas for the fat turkey, the probability drops to 0.7.

The example can be modeled in $p\mathcal{BC}+$ as follows. First, we declare the constants:

Notation: t range over $\{SlimTurkey, FatTurkey\}$.

Regular fluent constants:	Domains:
$Alive(t)$, $Loaded$	Boolean
Statically determined fluent constants:	Domains:
$Alert(t)$	Boolean
Action constants:	Domains:
$Load$, $Fire(t)$	Boolean
Pf constants:	Domains:
$Pf\_Killed(t)$, $Pf\_Killed\_Alert(t)$	Boolean
InitPf constants:	
$Init\_Alive(t)$, $Init\_Loaded$	Boolean

Next, we state the causal laws. The effect of loading the gun is described by

caused *Loaded* **if** $\top$ **after** *Load*.

To describe the effect of shooting at a turkey, we declare the following probability distributions on the result of shooting at each turkey when it is not alert and when it is alert, respectively:

caused $Pf\_Killed(SlimTurkey) = \{\mathbf{t} : 0.6, \mathbf{f} : 0.4\}$,

caused $Pf\_Killed(FatTurkey) = \{\mathbf{t} : 0.9, \mathbf{f} : 0.1\}$,

caused $Pf\_Killed\_Alert(SlimTurkey) = \{\mathbf{t} : 0.3, \mathbf{f} : 0.7\}$,

caused $Pf\_Killed\_Alert(FatTurkey) = \{\mathbf{t} : 0.7, \mathbf{f} : 0.3\}$.

The effect of shooting at a turkey is described as

$$\textbf{caused } \sim Alive(t) \textbf{ if } \top \textbf{ after } Loaded \wedge Fire(t) \wedge \sim Alert(t) \wedge Pf\_Killed(t),$$

$$\textbf{caused } \sim Alive(t) \textbf{ if } \top \textbf{ after } Loaded \wedge Fire(t) \wedge Alert(t) \wedge Pf\_Killed\_Alert(t),$$

$$\textbf{caused } \sim Loaded \textbf{ if } \top \textbf{ after } Fire(t).$$

A dead turkey causes the other turkey to be alert:

$$\textbf{default } \sim Alert(t),$$

$$\textbf{caused } Alert(t_1) \textbf{ if } \sim Alive(t_2) \wedge Alive(t_1) \wedge t_1 \neq t_2.$$

(**default** F stands for **caused** $\{F\}^{\text{ch}}$ Babb and Lee (2015)).

The fluents *Alive* and *Loaded* observe the commonsense law of inertia:

$$\textbf{caused } \{Alive(t)\}^{\text{ch}} \textbf{ if } \top \textbf{ after } Alive(t),$$

$$\textbf{caused } \{\sim Alive(t)\}^{\text{ch}} \textbf{ if } \top \textbf{ after } \sim Alive(t),$$

$$\textbf{caused } \{Loaded\}^{\text{ch}} \textbf{ if } \top \textbf{ after } Loaded,$$

$$\textbf{caused } \{\sim Loaded\}^{\text{ch}} \textbf{ if } \top \textbf{ after } \sim Loaded.$$

We ensure no concurrent actions are allowed by stating

$$\textbf{caused } \bot \textbf{ after } a_1 \wedge a_2$$

for every pair of action constants a_1, a_2 such that $a_1 \neq a_2$.

Finally, we state that the initial values of all fluents are uniformly random (b is a schematic variable that ranges over $\{\mathbf{t}, \mathbf{f}\}$):

$$\textbf{caused } Init\_Alive(t) = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\},$$

$$\textbf{caused } Init\_Loaded = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\},$$

$$\textbf{initially } Alive(t) = b \textbf{ if } Init\_Alive(t) = b,$$

$$\textbf{initially } Loaded = b \textbf{ if } Init\_Loaded = b.$$

We translate the action description into an LP$^{\text{MLN}}$ program and use LPMLN2ASP to answer various queries about transition systems, such as prediction, postdiction and planning queries.

Prediction For a prediction query, we are given a sequence of actions and observations that occurred in the past, and we are interested in the probability of a certain proposition describing the result of the history, or the most probable result of the history. Formally, we are interested in the conditional probability

$$Pr_{Tr(D,m)}(Result \mid Act, Obs)$$

or the MAP state

$$\underset{Result}{\text{argmax}} Pr_{Tr(D,m)}(Result \mid Act, Obs)$$

where $Result$ is a proposition describing a possible outcome, Act is a set of facts of the form $i : a$ or $i : \sim a$ for $a \in \sigma^{act}$, and Obs is a set of facts of the form $i : c = v$ for $c \in \sigma^{fl}$ and $v \in Dom(c)$.

In the Yale shooting example, such a query could be "given that only the fat turkey is alive and the gun is loaded at the beginning, what is the probability that the fat turkey died after shooting is executed?" To answer this query, we manually translate the action description above into the input language of LPMLN2ASP and add the following action and observation as constraints:

```
:- not alive(slimTurkey, f, 0).
:- not alive(fatTurkey, t, 0).
:- not loaded(t, 0).
:- not fire(fatTurkey, t, 0).
```

Executing the command

```
lpmln2asp -i yale-shooting.lpmln -q alive
```

yields

```
alive(fatTurkey, f, 1) 0.700000449318
```

Postdiction In the case of postdiction, we infer a condition about the initial state given the history. Formally, we are interested in the conditional probability

$$Pr_{Tr(D,m)}(Initial\_State \mid Act, Obs)$$

or the MAP state

$$\underset{Initial\_State}{\text{argmax}}\ Pr_{Tr(D,m)}(Initial\_State \mid Act, Obs)$$

where $Initial\_State$ is a proposition about the initial state; Act and Obs are defined as above.

In the Yale shooting example, such a query could be "given that the slim turkey was alive and the gun was loaded at the beginning, the person shot at the slim turkey and it died, what is the probability that the fat turkey was alive at the beginning?"

Formalizing the query and executing the command

```
lpmln2asp -i yale-shooting.lpmln -q alive
```

yields

```
alive(fatTurkey, t, 0) 0.666661211973
```

7.4 Diagnosis in Probabilistic Action Domains

One interesting type of reasoning tasks in action domains is diagnosis, where we observe a sequence of actions that fails to achieve some expected outcome and we would like to know possible explanations for the failure. Furthermore, in a probabilistic setting, we could also be interested in the probability of each possible explanation. In this section, we discuss how diagnosis can be automated in $p\mathcal{BC}+$ as probabilistic abduction and we illustrate the method through an example.

We define the following new constructs to allow probabilistic diagnosis in action domains. Note that those constructs are simply syntactic sugars that do not change the actual expressivity of the language.

- We introduce a subclass of regular fluent constants called *abnormal fluents*.
- When the action domain contains at least one abnormal fluent, we introduce a special statically determined fluent constant ab with Boolean domain, and we add
  default $\sim ab$.
- We introduce the expression

  $$\textbf{caused\_ab } F \textbf{ if } G \textbf{ after } H$$

  where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants and H does not contain initpf constants. This expression is treated as an abbreviation of

  $$\textbf{caused } F \textbf{ if } ab \wedge G \textbf{ after } H.$$

Once we have defined abnormalities and how they affect the system, we can use

$$\textbf{caused } ab$$

to enable taking abnormalities into account in reasoning.

The robot example introduced in Section 6.3.3 can be modeled in $p\mathcal{BC}+$ as follows. We first introduce the following constants:

Notation: r range over $\{R_1, R_2\}$.

Regular fluent constants: Domains:

$\mathit{LocRobot}$, $\mathit{LocBook}$	$\{R_1, R_2\}$
$\mathit{HasBook}$	Boolean
Abnormal fluent constants:	Domains:
$\mathit{EnterFailed}$, $\mathit{DropBook}$, $\mathit{PickupFailed}$	Boolean
Action constants:	Domains:
$\mathit{Goto}(r)$, $\mathit{PickUpBook}$, $\mathit{PutdownBook}$	Boolean
Pf constants:	Domains:
$\mathit{Pf\_EnterFailed}$, $\mathit{Pf\_PickupFailed}$, $\mathit{Pf\_DropBook}$	Boolean
Initpf constants:	Domains:
$\mathit{Init\_LocRobot}$, $\mathit{Init\_LocBook}$	$\{R_1, R_2\}$
$\mathit{Init\_HasBook}$	Boolean

The action $\mathit{Goto}(r)$ causes the location of the robot to be at r unless the abnormality $\mathit{EnterFailed}$ occurs:

caused $\mathit{LocRobot}=r$ **after** $\mathit{Goto}(r) \wedge \neg \mathit{EnterFailed}$.

Similarly, the following causal laws describe the effect of the actions $\mathit{PickupBook}$ and $\mathit{PutdownBook}$:

caused $\mathit{HasBook}$ **if** $\mathit{LocRobot} = \mathit{LocBook}$ **after** $\mathit{PickUpBook} \wedge \neg \mathit{PickUpFailed}$
caused $\sim\mathit{HasBook}$ **after** $\mathit{PutdownBook}$.

If the robot has the book, then the book has the same location as the robot:

caused $\mathit{LocBook} = r$ **if** $\mathit{LocRobot} = r \wedge \mathit{HasBook}$.

The abnormality $\mathit{DropBook}$ causes the robot to not have the book:

caused $\sim\mathit{HasBook}$ **if** $\mathit{DropBook}$.

The fluents *LocBook*, *LocRobot* and *HasBook* observe the commonsense law of inertia:

$$\textbf{caused } \{LocBook = r\}^{\text{ch}} \textbf{ after } LocBook = r \qquad \textbf{caused } \{LocRobot = r\}^{\text{ch}} \textbf{ after } LocRobot = r$$
$$\textbf{caused } \{HasBook = b\}^{\text{ch}} \textbf{ after } LocBook = b.$$

The abnormality *EnterFailed* has 0.2 chance to occur when the action *Goto* is executed:

$$\textbf{caused } \{\sim EnterFailed\}^{\text{ch}} \textbf{ if } \sim EnterFailed \qquad \textbf{caused } Pf\_EnterFailed = \{\mathbf{t} : 0.2, \mathbf{f} : 0.8\}$$
$$\textbf{caused\_ab } EnterFailed \textbf{ if } \top \textbf{ after } pf\_EnterFailed \wedge Goto(r).$$

Similarly, the following causal laws describe the condition and probabilities for the abnormalities *PickupFailed* and *DropBook* to occur:

$$\textbf{caused } \{\sim PickupFailed\}^{\text{ch}} \textbf{ if } \sim PickupFailed \qquad \textbf{caused } Pf\_PickupFailed = \{\mathbf{t} : 0.3, \mathbf{f} : 0.7\}$$
$$\textbf{caused\_ab } PickupFailed \textbf{ if } \top \textbf{ after } Pf\_PickupFailed \wedge PickupBook,$$

$$\textbf{caused } \{\sim DropBook\}^{\text{ch}} \textbf{ if } \sim DropBook \qquad \textbf{caused } Pf\_DropBook = \{\mathbf{t} : 0.1, \mathbf{f} : 0.9\}$$
$$\textbf{caused\_ab } DropBook \textbf{ if } \top \textbf{ after } Pf\_DropBook \wedge HasBook.$$

We ensure no concurrent actions are allowed by stating

$$\textbf{caused } \bot \textbf{ after } a_1 \wedge a_2$$

for every pair of action constants a_1, a_2 such that $a_1 \neq a_2$. Initially, it is uniformly random where the robot and the book is and whether the robot has the book:

$$\textbf{caused } Init\_LocRobot = \{R_1 : 0.5, R_2 : 0.5\} \qquad \textbf{initially } LocRobot = r \textbf{ if } Init\_LocRobot = r$$
$$\textbf{caused } Init\_LocBook = \{R_1 : 0.5, R_2 : 0.5\} \qquad \textbf{initially } LocBook = r \textbf{ if } Init\_LocBook = r$$
$$\textbf{caused } Init\_HasBook = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\}, \qquad \textbf{initially } HasBook = b \textbf{ if } Init\_HasBook = b.$$

No abnormalities are possible at the initial state:

initially $\bot$ **if** $EnterFailed$, **initially** $\bot$ **if** $PickupFailed$, **initially** $\bot$ **if** $DropBook$.

We add **caused** ab to the action description to take abnormalities into account in reasoning and translate the action description into LP$^{\text{MLN}}$ program.

Executing `lpmln2asp -i robot.lpmln` yields

```
pickupBook("t",0)  ab("pickup_failed","t",1)  goto("r2","t",1) putdownBook("t",2)
```

which suggests that the robot fails at picking up the book.

Suppose that the robot has observed that the book was in its hand after it picked up the book. We expand the action history with

```
:- not hasBook("t", 1).
```

Now the most probable stable model becomes

```
pickupBook("t",0)  goto("r2","t",1)  ab("drop_book","t",2)  putdownBook("t",2)
```

suggesting that robot accidentally dropped the book.

On the other hand, if the robot further observed that itself was not at `r2` after the execution

```
:- locRobot("r2", 3).
```

Then the most probable stable model becomes

```
pickupBook("t",0)  goto("r2","t",1)  ab("enter_failed","t",2)  putdownBook("t",2)
```

suggesting that the robot failed at entering `r2`.

7.5 Related Work

There exist various formalisms for reasoning in probabilistic action domains. $P\mathcal{C}+$ (Eiter and Lukasiewicz (2003)) is a generalization of the action language $\mathcal{C}+$ that allows for expressing probabilistic information. The syntax of $P\mathcal{C}+$ is similar to $p\mathcal{BC}+$, as both the languages are extensions of $\mathcal{C}+$. $P\mathcal{C}+$ expresses probabilistic transition of states through so-called *context variables*, which are similar to pf constants in $p\mathcal{BC}+$, in that they are both exogenous variables associated with predefined probability distributions. In $p\mathcal{BC}+$, in order to achieve meaningful probability computed through LP^{MLN}, assumptions such as all actions have to be always executable and nondeterminism can only be caused by pf constants, have to be made. In contrast, $P\mathcal{C}+$ does not impose such semantic restrictions, and allows for expressing qualitative and quantitative uncertainty about actions by referring to the sequence of "belief" states—possible sets of states together with probabilistic information.

On the other hand, the semantics is highly complex and there is no implementation of $P\mathcal{C}+$ as far as we know. Zhu (2012) defined a probabilistic action language called $\mathcal{NB}$, which is an extension of the (deterministic) action language $\mathcal{B}$. $\mathcal{NB}$ can be translated into P-log (Baral *et al.* (2004)) and since there exists a system for computing P-log, reasoning in $\mathcal{NB}$ action descriptions can be automated. Like $p\mathcal{BC}+$ and $P\mathcal{C}+$, probabilistic transitions are expressed through dynamic causal laws with random variables associated with predefined probability distribution. In $\mathcal{NB}$, however, these random variables are hidden from the action description and are only visible in the translated P-log representation. One difference between $\mathcal{NB}$ and $p\mathcal{BC}+$ is that in $\mathcal{NB}$ a dynamic causal law must be associated with an action and thus can only be used to represent probabilistic effect of actions, while in $p\mathcal{BC}+$, a fluent dynamic law can have no action constant occurring in it. This means state transition without actions or time step change cannot be expressed directly in $\mathcal{NB}$. Like p$\mathcal{BC}+$, in order to translate $\mathcal{NB}$ into executable low-level logic programming languages, some semantical assumptions have to be made in $\mathcal{NB}$. The assumptions made in $\mathcal{NB}$ are very similar to the ones made in $p\mathcal{BC}+$.

Probabilistic action domains, especially in terms of probabilistic effects of actions, can be formalized as Markov Decision Process (MDP). The language proposed in Baral *et al.* (2002) aims at facilitating elaboration tolerant representations of MDPs. The syntax is similar to $p\mathcal{BC}+$. The semantics is more complex as it allows preconditions of actions and imposes less semantical assumption. The concept of *unknown variables* associated with probability distributions is similar to pf constants in our setting. There is, as far as we know, no implementation of the language. There is no discussion about probabilistic diagnosis in the context of the language. PPDDL Younes and Littman (2004) is a probabilistic extension of the planning definition language PDDL. Like $\mathcal{NB}$, the nondeterminism that PPDDL considers is only the probabilistic effect of actions. The semantics of PPDDL is defined in terms of MDP. There are also probabilistic extensions of the Event Calculus such as D'Asaro *et al.* (2017) and Skarlatidis *et al.* (2011).

In the above formalisms, the problem of probabilistic diagnosis is only discussed in Zhu (2012). Balduccini and Gelfond (2003) and Baral *et al.* (2000) studied the problem of diagnosis. However, they are focused on diagnosis in deterministic and static domains. Iwan (2002) has proposed a method for diagnosis in action domains with situation calculus. Again, the diagnosis considered there does not involve any probabilistic measure.

Compared to the formalisms mentioned here, the unique advantages of p$\mathcal{BC}$+ include its executability through LP$^{\text{MLN}}$ systems, its support for probabilistic diagnosis, and the possibility of parameter learning in actions domains.

7.6 Proofs

7.6.1 Proof of Proposition 11

Proposition 11 *For any transition $\langle s, e, s' \rangle$, s and s' are states.*

Proof. To show that s and s' are states, we show that $0 : s$ and $0 : s'$ are stable models of D_0. We use I^0 as an abbreviation of $0:s \cup 0:e \cup 1:s' \cup 0:pf$. By definition of a transition, we know that $0:s \cup 0:e \cup 1:s'$ is a residual stable model of D_1, i.e., there exists an assignment pf to σ^{pf} such that $0:s \cup 0:e \cup 1:s' \cup 0:pf$ is a stable model of D_1. By definition of a probabilistic stable model, this means $0:s \cup 0:e \cup 1:s' \cup 0:pf$ is a (deterministic) stable model of

$$SD(0) \cup FD(0) \cup \overline{PF(0)_{I^0}} \cup UEC \cup EXG. \tag{7.10}$$

$0:s$ **is a stable model of** D_0: We split (7.10) into two disjoint subsets

$$SD(0)$$

and

$$SD(1) \cup FD(0) \cup \overline{PF(0)_{I^0}} \cup UEC \cup EXG.$$

It can be seen that $SD(0)$ is negative on $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$ and $SD(1) \cup FD(0) \cup \overline{PF(0)_{I^0}} \cup UEC \cup EXG$ is negative on $0:\sigma^{fl}$. Every strongly connected components of (7.10) is either a subset of $0:\sigma^{fl}$ or a subset of $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$. By the splitting theorem, we have that $0:s$ is stable model of $SD(0)$ w.r.t. $0:\sigma^{fl}$ and $0:e \cup 1:s' \cup 0:pf$ is a stable model of $SD(1) \cup FD(0) \cup \overline{PF(0)_{I^0}} \cup UEC \cup EXG$ w.r.t. $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$. Since $D_0 = SD(0)$, s' is a stable model of D_0.

$0:s'$ **is a stable model of** D_0: We further divide the set of fluents into the set σ^r of regular fluents and the set σ^{sd} of statically determined fluents. From the above reasoning, we know that $0:e \cup 1:s' \cup 0:pf$ is a stable model of

$$SD(1) \cup FD(0) \cup \overline{PF(0)_{I^0}} \cup UEC \cup EXG$$

w.r.t. $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$, i.e. $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{r} \cup 1:\sigma^{sd}$, . By Theorem 2 in Ferraris *et al.* (2009), we have that $0:e \cup 1:s' \cup 0:pf$ is a stable model of

$$SD(1) \cup FD(0) \cup \overline{PF(0)_{I^0}} \cup UEC \cup EXG$$

w.r.t. $1 : \sigma^{sd}$. Since $FD(0)$, $PF(0)$, UEC and EXG are negative on $1 : \sigma^{sd}$, this implies $0 : e \cup 1 : s' \cup 0 : pf$ is a stable model of $SD(1)$ w.r.t. $1 : \sigma^{sd}$. Since $SD(1)$ does not mention any atom in $0 : e \cup 0 : pf$, we have that $1 : s'$ is a stable model of $SD(1)$ w.r.t. $1 : \sigma^{sd}$. Let $(1 : \sigma^r)^{\text{ch}}$ denote the set of rules of the form (7.7) for each $c \in \sigma^r$. The above implies that $1 : s'$ is a stable model of $SD(1) \cup (1 : \sigma^r)^{\text{ch}}$ w.r.t. $1 : \sigma^{sd} \cup 1 : \sigma^r = 1 : \sigma^{fl}$. Changing all the timesteps from 1 to 0, we obtain that $0 : s'$ is a stable model of $SD(0) \cup (0 : \sigma^r)^{\text{ch}} = D_0$ w.r.t. $0 : \sigma^{fl}$. $\square$

7.6.2 Proof of Theorem 21

Proposition 12. *For any multi-valued probabilistic program* Π *for which every total choice leads to* $n (n > 0)$ *stable models and any interpretation* I*, we have*

$$P''_{\Pi}(I) = \frac{1}{n} W''_{\Pi}(I).$$

Proof. We show that the normalization factor is constant n, i.e.,

$$\sum_{I \text{ is an interpretation of } \Pi} W''_{\Pi}(I) = n.$$

Let $pf_1, \dots, pf_m$ be the probabilistic constants in Π, and $v_{i,1}, \dots, v_{i,k_i}$, each associated with probability $p_{i,1}, \dots, p_{i,k_i}$ resp. be the values of pf_i ($i \in \{1, \dots, m\}$). Let TC_{Π} be the set of all assignments to probabilistic constants in Π.

$$
\begin{aligned}
&\sum_{I \text{ is an interpretation of } \Pi} W''_{\Pi}(I)\\
=&\sum_{I\in SM''[\Pi]} W''_{\Pi}(I)\\
=&\sum_{tc\in TC_{\Pi}} n\cdot \prod_{c=v\in tc} M_{\Pi}(c=v)\\
=&n\sum_{tc\in TC_{\Pi}}\prod_{c=v\in tc} M_{\Pi}(c=v)\\
=&n\cdot (p_{1,1}\sum_{\substack{tc\in TC_{\Pi}\\ tc(pf_1)=v_{1,1}}}\prod_{\substack{c=v\in tc\\ c\neq pf_1}} M_{\Pi}(c=v)+\cdots+p_{1,k_1}\sum_{\substack{tc\in TC_{\Pi}\\ tc(pf_1)=v_{1,k_1}}}\prod_{\substack{c=v\in tc\\ c\neq pf_1}} M_{\Pi}(c=v))\\
=&n\cdot (\sum_{i_1\in\{1,\dots,k_1\}} p_{1,i_1}\sum_{\substack{tc\in TC_{\Pi}\\ tc(pf_1)=v_{1,i_1}}}\prod_{\substack{c=v\in tc\\ c\neq pf_1}} M_{\Pi}(c=v))\\
=&n\cdot (\sum_{i_2\in\{1,\dots,k_2\}} p_{2,i_2}\sum_{i_1\in\{1,\dots,k_1\}} p_{1,i_1}\sum_{\substack{tc\in TC_{\Pi}\\ tc(pf_1)=v_{1,i_1}\\ tcpf_2=v_{2,i_2}}}\prod_{\substack{c=v\in tc\\ c\neq pf_1\\ c\neq pf_2}} M_{\Pi}(c=v))\\
=&n\cdot (\sum_{i_{m-1}\in\{1,\dots,k_{m-1}\}} p_{m,i_m}\cdots\sum_{i_2\in\{1,\dots,k_2\}} p_{2,i_2}\sum_{i_1\in\{1,\dots,k_1\}} p_{1,i_1}\\
&\qquad\qquad \sum_{\substack{tc\in TC_{\Pi}\\ tc(pf_1)=v_{1,i_1}\\ tc(pf_2)=v_{2,i_2}\\ \cdots\\ tc(pf_{m-1})=v_{m-1,i_m}}}\prod_{\substack{c=v\in tc\\ c\neq pf_1\\ c\neq pf_2\\ \cdots\\ c\neq pf_{m-1}}} M_{\Pi}(c=v))\\
=&n\cdot (\sum_{i_{m-1}\in\{1,\dots,k_{m-1}\}} p_{m,i_m}\cdots\sum_{i_2\in\{1,\dots,k_2\}} p_{2,i_2}\sum_{i_1\in\{1,\dots,k_1\}}\\
&\qquad\qquad p_{1,i_1}(M_{\Pi}(pf_m=v_{m,1})+\cdots+M_{\Pi}(pf_m=v_{m,k_m})))\\
=&n\cdot (\sum_{i_{m-1}\in\{1,\dots,k_{m-1}\}} p_{m,i_m}\cdots\sum_{i_2\in\{1,\dots,k_2\}} p_{2,i_2}\sum_{i_1\in\{1,\dots,k_1\}} p_{1,i_1}1)\\
=&n\cdot (1)\\
=&n.
\end{aligned}
$$

$\square$

For $i' \in \{0, \ldots, m\}$, we use $SD(i')$ to denote the set of all rules of the form (7.5) in D_m where $i = i'$, and for $i' \in \{0, \ldots, m-1\}$, we use $FD(i')$ to denote the set of all rules of the form (7.6) in D_m where $i = i'$. Furthermore, according to Lee and Wang (2016), a pf constant declaration (7.9) is translated into

$$ln(p_j) \quad : \quad (i : pf) = v_j \tag{7.11}$$

for each $j \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, m-1\}$. For any $i \in \{0, \ldots, m-1\}$, and any assignment to σ^{pf}, we use $PF(i)$ to denote the set of weighted rules (7.11) in D_m where pf is an pf constant, and $PF(i)_{TC}$ to denote the subset of $PF(i)$ that TC satisfies.

Also, from the definition of multi-valued probabilistic programs (3.3), D_m contains

$$\alpha : \quad \bot \leftarrow c = v_1 \wedge c = v_2 \tag{7.12}$$

and

$$\alpha : \quad \bot \leftarrow \neg \bigvee_{v \in Dom(c)} c = v \tag{7.13}$$

for all constants c and $v_1, v_2 \in Dom(c)$ such that $v_1 \neq v_2$. We use UEC to denote the set of rules of the form (7.12) or (7.13), and EXG to denote the set of rules of the form (7.8) and (7.7), disregarding their weights.

Lemma 21. *Let $(\langle s, a, s' \rangle, pf)$ be a pf-transition of D. We have that $0 : s \cup 0 : a \cup 0 : pf \cup 1 : s'$ is a (deterministic) stable model of $SD(1) \cup FD(0) \cup \overline{PF(0)}_{0:pf} \cup UEC$ w.r.t. $0 : \sigma^{act} \cup 0 : \sigma^{pf} \cup 1 : \sigma^{fl}$.*

Proof. By definition of pf-transition, we have that I is a deterministic stable model of

$$SD(0) \cup SD(1) \cup FD(0) \cup \overline{PF(0)_I} \cup UEC. \tag{7.14}$$

We split (7.14) into $SD(0)$ and the rest

$$SD(1) \cup FD(0) \cup \overline{PF(0)_I} \cup UEC.$$

It can be seen that $SD(0)$ is negative on $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$ and $SD(1) \cup FD(0) \cup \overline{PF(0)_I} \cup UEC$ is negative on $0:\sigma^{fl}$. Each strongly connected components of (7.14) is either a subset of $0:\sigma^{fl}$ or a subset of $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$.

By the splitting theorem, we have that $0:s \cup 0:a \cup 0:pf \cup 1:s'$ is a (deterministic) stable model of $SD(1) \cup FD(0) \cup \overline{PF(0)}_{0:pf} \cup UEC$ w.r.t. $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$.

□

For any set of constants $\mathcal{C}$, a of $\mathcal{C}$ is a function that maps each element c in $\mathcal{C}$ to a unique element in $Dom(c)$. We say an interpretation I of the propositional signature constructed from $\mathcal{C}$ (as described in Section 3.3) *satisfies* a value assignment V of $\mathcal{C}$ if for all $c \in \mathcal{C}$, $(c = v)^I = \mathbf{t}$ if and only if $V(c) = v$.

Theorem 21 *Given any value assignment TC of constants in $\sigma_m^{pf} \cup 0:\sigma^{initpf}$ and a value assignment A of constants of σ_m^{act}, $I_{TC\cup A}$ is the only stable model of $Tr(D,m)$ that satisfies $TC \cup A$, and the probability of $I_{TC\cup A}$ is*

$$Pr_{Tr(D,m)}(I_{TC\cup A}) = \frac{\prod\limits_{c=v\in TC} M(c=v)}{(|\sigma^{act}|+1)^m}.$$

Proof. We first show that $I_{TC\cup A}$ is the only stable model of $Tr(D,m)$ that satisfies $TC \cup A$. Clearly $I_{TC\cup A} \models TC \cup A$. We use $I^i_{TC\cup A}$ for $i \in \{0,1,\dots,m-1\}$ to denote the following subset of $I_{TC\cup A}$:

$$(I_{TC\cup A})|_{i:\sigma^{fl}} \cup (I_{TC\cup A})|_{i:\sigma^{act}} \cup (I_{TC\cup A})|_{i+1:\sigma^{fl}} \cup (I_{TC\cup A})|_{i:\sigma^{pf}}.$$

For any $i, j \in \{0,\dots,m\}$ such that $i < j$ and any signature σ', we use $i..j:\sigma'$ to denote $i:\sigma' \cup \cdots \cup j:\sigma'$.

- **We show that $I_{TC\cup A}$, i.e., $(I_{TC\cup A})|_{0:\sigma^{initpf}} \cup I^0_{TC\cup A} \cup \cdots \cup I^{m-1}_{TC\cup A}$ is a probabilistic stable model of $Tr(D,m)$ by induction:** Let $I_{TC\cup A}(n)$ denote $(I_{TC\cup A})|_{0:\sigma^{initpf}} \cup I^0_{TC\cup A} \cup \cdots \cup I^{n-1}_{TC\cup A}$

  Base Case: when $m = 1$, consider $I_{TC\cup A}(1)$, i.e, $(I_{TC\cup A})|_{0:\sigma^{initpf}} \cup I^0_{TC\cup A}$.

$$\overline{Tr(D,1)_{I_{TC\cup A}(1)}}$$

  is the ASP program

$$\begin{aligned}
&\overline{(D_{init})_{I_{TC\cup A}(1)}}\cup \\
&SD(0) \cup SD(1) \cup \\
&FD(0) \cup \\
&\overline{PF(0)_{TC}} \cup UEC.
\end{aligned}$$

  Since

$$(\langle (I_{TC\cup A})|_{0:\sigma^{fl}}, (I_{TC\cup A})|_{0:\sigma^{act}}, (I_{TC\cup A})|_{1:\sigma^{fl}}\rangle, (I_{TC\cup A})|_{0:\sigma^{pf}})$$

  is a pf-transition, by Lemma 21, $I_{TC\cup A}(1)$ is a deterministic stable model of $SD(1) \cup FD(0) \cup \overline{PF(0)_{TC}} \cup UEC$ w.r.t. $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$.

  On the other hand, from the construction of $I_{TC\cup A}$, $I_{TC\cup A}(1)$ is a deterministic stable model of $\overline{(D_{init})_{(I_{TC\cup A})}} \cup SD(0)$ w.r.t. $0:\sigma^{fl} \cup 0:\sigma^{initpf}$.

  It can be seen that $SD(1) \cup FD(0) \cup \overline{PF(0)_{TC}} \cup UEC$ is negative on $0:\sigma^{fl} \cup 0:\sigma^{initpf}$ and $\overline{(D_{init})_{(I_{TC\cup A})}} \cup SD(0)$ is negative on $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$. Each strongly component of the dependency graph of $(Tr(D,m))_{I_{TC\cup A}(1)}$ is either a subset of $0:\sigma^{fl} \cup 0:\sigma^{initpf}$ or a subset of $0:\sigma^{act} \cup 0:\sigma^{pf} \cup 1:\sigma^{fl}$.

  Applying the splitting theorem, we have that $I_{TC\cup A}(1)$ is a deterministic stable model of $\overline{(Tr(D,m))_{I_{TC\cup A}(1)}}$ and thus is a probabilistic stable model of $Tr(D,m)$, since it does not violate any hard rules.

For $m > 1$, consider $I_{TC \cup A}(m)$.

$$\begin{aligned}\overline{(Tr(D,m))_{I_{TC\cup A}(m)}} = {} & \overline{(D_{init})_{I_{TC\cup A}(m)}} \cup \\ & SD(0) \cup \cdots \cup SD(m) \cup \\ & FD(0) \cup \cdots \cup FD(m-1) \cup \\ & \overline{PF(0)_{TC}} \cup \cdots \cup \overline{PF(m-1)_{TC}} \cup UEC\end{aligned} \tag{7.15}$$

Clearly we have

- each strongly connected component of the dependency graph of (7.15) is either a subset of $0 : \sigma^{initpf} \cup 0..(m-1) : \sigma^{fl} \cup 0..(m-2) : \sigma^{act} \cup 0..(m-2) : \sigma^{pf}$ or a subset of $m : \sigma^{fl} \cup m-1 : \sigma^{act} \cup m-1 : \sigma^{pf}$;
-

$$\begin{aligned}\overline{(Tr(D,m-1))_{I_{TC\cup A}(m)}} = {} & \overline{(D_{init})_{I_{TC\cup A}(m)}} \cup \\ & SD(0) \cup \cdots \cup SD(m-1) \cup \\ & FD(0) \cup \cdots \cup FD(m-2) \cup \\ & \overline{PF(0)_{TC}} \cup \cdots \cup \overline{PF(m-2)_{TC}} \cup UEC \cup EXG\end{aligned}$$

  is negative on $m : \sigma^{fl} \cup m-1 : \sigma^{act} \cup m-1 : \sigma^{pf}$;
-

$$\begin{aligned}&\overline{(Tr(D,m) \setminus Tr(D,m-1))_{I_{TC\cup A}(m)}} = \\ &SD(m) \cup \\ &FD(m-1) \cup \\ &\overline{PF(m-1)_{TC}}\end{aligned}$$

  is negative on $0 : \sigma^{initpf} \cup 0..m-1 : \sigma^{fl} \cup 0..m-2 : \sigma^{act} \cup 0..m-2 : \sigma^{pf}$.

By I.H., $I_{TC\cup A}(m-1)$ is a probabilistic stable model of $Tr(D, m-1)$, which implies $I_{TC\cup A}(m)$ is a (deterministic) stable model of

$$\overline{(Tr(D, m-1))_{I_{TC\cup A}(m)}}$$

w.r.t. $0 : \sigma^{initpf} \cup 0..m-1 : \sigma^{fl} \cup 0..m-2 : \sigma^{act} \cup 0..m-2 : \sigma^{pf}$. The fact that $I_{TC\cup A}(m)$ is a (deterministic) stable model of

$$\overline{(Tr(D, m) \setminus Tr(D, m-1))_{I_{TC\cup A}(m)}}$$

w.r.t. $m : \sigma^{fl} \cup m-1 : \sigma^{act} \cup m-1 : \sigma^{pf}$ can be seen from Lemma 21 and replacing timesteps m and $m-1$ with 1 and 0 resp.

Thus, $I_{TC\cup A}(m)$ is a stable model of $Tr(D, m)$.

- **There does not exist more than one stable models of $Tr(D, m)$ that satisfies $TC \cup A$.** Suppose, to the contrary, that there exists $I \neq I_{TC\cup A}$ that satisfies $TC \cup A$ and I is also a stable model of $Tr(D, m)$. Since I and $I_{TC\cup A}$ agree on $TC \cup A$, they can differ only on the value assignment of constants in σ^{fl}. Let $i : fl$ be any one of the constants such that $I(i : fl) \neq I_{TC\cup A}(i : fl)$ and there does not exist any $j : fl'$ with $j \leq i$ and $I(j : fl') \neq I_{TC\cup A}(j : fl')$. By definition, the assumption that I is a probabilistic stable model of $Tr(D, m)$ means I is a (deterministic) stable model of

$$\begin{aligned}\overline{Tr(D, m)}_I = \overline{D_{init}}_I \cup \\ & SD(0) \cup \cdots \cup SD(m) \cup \\ & FD(0) \cup \cdots \cup FD(m-1) \cup \\ & \overline{PF(0)_{TC}} \cup \cdots \cup \overline{PF(m-1)_{TC}} \cup \\ & UEC \cup EXG\end{aligned}$$

which, by the splitting theorem, implies that I is a stable model of

$$\begin{aligned}
&\overline{(D_{init})_I} \cup \\
&SD(0) \cup \cdots \cup SD(i-1) \cup SD(i+1) \cup \cdots \cup SD(m) \cup \\
&FD(0) \cup \cdots \cup FD(i-2) \cup FD(i) \cup \ldots FD(m-1) \cup \\
&\overline{PF(0)_{TC}} \cup \cdots \cup \overline{PF(i-2)_{TC}} \cup \overline{PF(i)_{TC}} \cup \cdots \cup \overline{PF(m-1)_{TC}} \cup \\
&\overline{INIT_{TC}} \cup UEC \cup EXG
\end{aligned}$$

w.r.t. $\sigma^{initpf} \cup 0..(i-1) : \sigma^{fl} \cup (i+1)..m : \sigma^{fl} \cup 0..(i-2) : \sigma^{act} \cup i..(m-1) : \sigma^{act} \cup 0..(i-2) : \sigma^{pf} \cup i..(m-1) : \sigma^{pf}$, and I is a stable model of

$$SD(i) \cup FD(i-1) \cup \overline{PF(i-1)_{TC}}$$

w.r.t. $i : \sigma^{fl} \cup i-1 : \sigma^{act} \cup i-1 : \sigma^{pf}$. Changing the timesteps, this means

$$0 : I|_{i-1:\sigma^{fl}}, 0 : I|_{i-1:\sigma^{act}}, 1 : I|_{i:\sigma^{fl}}, 0 : I|_{i-1:\sigma^{pf}}$$

is a stable model of

$$SD(1) \cup FD(0) \cup \overline{PF(0)_{TC}} \cup UEC \cup EXG$$

w.r.t. $1 : \sigma^{fl} \cup 0 : \sigma^{act} \cup 0 : \sigma^{pf}$. On the other hand, clearly, $0 : I|_{i-1:\sigma^{fl}}, 0 : I|_{i-1:\sigma^{act}}, 1 : I|_{i:\sigma^{fl}}, 0 : I|_{i-1:\sigma^{pf}}$ also satisfies $SD(0)$. Due to the existence of EXG, we have

$$0 : I|_{i-1:\sigma^{fl}}, 0 : I|_{i-1:\sigma^{act}}, 1 : I|_{i:\sigma^{fl}}, 0 : I|_{i-1:\sigma^{pf}}$$

is a stable model of

$$SD(0) \cup SD(1) \cup FD(0) \cup \overline{PF(0)_{TC}} \cup UEC \cup EXG = D_1$$

The above implies that $(\langle I|_{i-1:\sigma^{fl}}, I|_{i-1:\sigma^{act}}, I|_{i:\sigma^{fl}}\rangle, I|_{i-1:\sigma^{pf}})$ is also a pf-transition in addition to

$$(\langle (I_{TC\cup A})|_{i-1:\sigma^{fl}}, (I_{TC\cup A})|_{i-1:\sigma^{act}}, (I_{TC\cup A})|_{i:\sigma^{fl}}\rangle, (I_{TC\cup A})|_{i-1:\sigma^{pf}}),$$

which contradict our assumption 2.

Consequently, in $Tr(D, m)$, since there are $(|\sigma^{act}| + 1)^m$ different assignments of σ^{act} under Assumption 1, every total choice leads to $(|\sigma^{act}| + 1)^m$ stable models. By Theorem 12, we have

$$Pr_{Tr(D,m)}(I_{TC\cup A}) = \frac{\prod\limits_{c=v\in TC} M(c = v)}{(|\sigma^{act}| + 1)^m}.$$

□

7.6.3 Proof of Theorem 22 and Corollary 2

For a multi-valued probabilistic program $\mathbf{\Pi}$, a *total choice* of $\mathbf{\Pi}$ is a value assignment to probabilistic constants in $\mathbf{\Pi}$. For any interpretation I, of a multi-valued probabilistic program, that satisfies uniqueness and existence constraints for all constants, the *total choice* of I, denoted $TC(I)$, is the function that maps each probabilistic constant c to the value v such that $c = v \in I$. We say a total choice *tc leads* to an interpretation I if I satisfies *tc*.

In the following proofs, we sometimes identify a value assignment A with the set

$$\{c = v \mid A(c) = v\}.$$

Proposition 13. *For any multi-valued probabilistic program $\mathbf{\Pi} = \langle PF, \Pi\rangle$ such that every total choice leads to the same number of stable models, we have*

$$Pr_{\mathbf{\Pi}}(c = v) = M_{\mathbf{\Pi}}(c = v)$$

for any probabilistic constant c and $v \in Dom(c)$.

Proof. Let n be the number of stable models that each total choices leads to. By

Proposition 12 we have

$$
\begin{aligned}
&Pr_{\mathbf{\Pi}}(c=v)\\
=&\sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models c=v}} \frac{\prod\limits_{c'=v' \in TC(I)} M_{\mathbf{\Pi}}(c'=v')}{n}\\
=&M_{\mathbf{\Pi}}(c=v) \cdot \frac{1}{n} \cdot \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models c=v}} \prod_{\substack{c'=v' \in TC(I) \\ c' \neq c}} M_{\mathbf{\Pi}}(c'=v')\\
=&M_{\mathbf{\Pi}}(c=v) \cdot \frac{1}{n} \cdot n \sum_{v' \in Dom(c_1)} M_{\mathbf{\Pi}}(c_1=v') \dots \sum_{v' \in Dom(c_n)} M_{\mathbf{\Pi}}(c_n=v')\\
=&M_{\mathbf{\Pi}}(c=v) \cdot \frac{1}{n} \cdot n \cdot 1\\
=&M_{\mathbf{\Pi}}(c=v)
\end{aligned}
$$

□

Proposition 14. *For any multi-valued probabilistic program $\mathbf{\Pi} = \langle PF, \Pi \rangle$ such that every total choice leads to the same number of stable models, and any value assignment ppf of a subset P of probabilistic constants in $\mathbf{\Pi}$, we have*

$$Pr_{\mathbf{\Pi}}(\bigwedge_{pf=v \in ppf} pf=v) = \prod_{pf=v \in ppf} Pr_{\mathbf{\Pi}}(pf=v).$$

Proof. Let n denote the number of stable models each total choice leads to. By

Proposition 12 we have

$$
\begin{aligned}
& Pr_{\mathbf{\Pi}}(\bigwedge_{ppf(pf)=v} pf = v) \\
= & \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models \bigwedge\limits_{ppf(pf)=v} pf=v}} \frac{W''_{\mathbf{\Pi}}(I)}{n} \\
= & \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models \bigwedge\limits_{ppf(pf)=v} pf=v}} \frac{\prod\limits_{c=v \in TC(I)} M_{\mathbf{\Pi}}(c=v)}{n} \\
= & \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models \bigwedge\limits_{ppf(pf)=v} pf=v}} \frac{\prod\limits_{ppf(pf)=v} M_{\mathbf{\Pi}}(pf=v) \cdot \prod\limits_{c=v \in TC(I) \setminus ppf} M_{\mathbf{\Pi}}(c=v)}{n} \\
= & \prod_{ppf(pf)=v} M_{\mathbf{\Pi}}(pf=v) \cdot \frac{1}{n} \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models \bigwedge\limits_{ppf(pf)=v} pf=v}} \prod_{c=v \in TC(I) \setminus ppf} M_{\mathbf{\Pi}}(c=v)
\end{aligned}
$$

We use C to denote the set of all constants in $\mathbf{\Pi}$. Let $C \setminus P = \{c_1, \dots, c_n\}$. Since every total choice leads to the same number of stable models, we have

$$
\begin{aligned}
& \frac{1}{n} \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I \models \bigwedge\limits_{pf=v \in ppf} pf=v}} \prod_{c=v \in TC(I) \setminus ppf} M_{\mathbf{\Pi}}(c=v) \\
= & \frac{1}{n} \sum_{TC \text{ is a value assignment of } C \setminus P} n \cdot \prod_{TC(c)=v} M_{\mathbf{\Pi}}(c=v) \\
= & \frac{1}{n} \cdot n \sum_{v \in Dom(c_1)} M_{\mathbf{\Pi}}(c_1 = v) \dots \sum_{v \in Dom(c_n)} M_{\mathbf{\Pi}}(c_n = v) \\
= & 1.
\end{aligned}
$$

Consequently by Proposition 13 we have

$$\begin{aligned}
&Pr_{\mathbf{\Pi}}(\bigwedge_{pf=v\in ppf} pf=v) \\
&= \prod_{pf=v\in ppf} M_{\mathbf{\Pi}}(pf=v)\cdot \frac{1}{n} \sum_{\substack{I \text{ is a stable model of } \mathbf{\Pi} \\ I\models \bigwedge_{pf=v\in ppf} pf=v}} \prod_{c=v\in TC(I)\setminus ppf} M_{\mathbf{\Pi}}(c=v) \\
&= \prod_{pf=v\in ppf} M_{\mathbf{\Pi}}(pf=v)\cdot 1 \\
&= \prod_{pf=v\in ppf} Pr_{\mathbf{\Pi}}(pf=v).
\end{aligned}$$

□

Theorem 22 *For any state s and s', and action e, we have*

$$Pr_{Tr(D,m)}(i+1:s' \mid i:s, i:e) = Pr_{Tr(D,m)}(j+1:s' \mid j:s, j:e)$$

for any $i,j\in\{0,\dots,m-1\}$ such that $Pr_{Tr(D,m)}(i:s)\neq 0$ and $Pr_{Tr(D,m)}(j:s)\neq 0$.

Proof. For any $k\in\{0,\dots,m-1\}$ such that $Pr_{Tr(D,m)}(k:s)\neq 0$, we show that

$$Pr_{Tr(D,m)}(k+1:s' \mid k:s, k:e) = Pr_{D_m}(1:s' \mid 0:s, 0:e).$$

Firstly, since $Tr(D,m)$ satisfies the condition that every total choice leads to the same number of stable models, by Proposition 13, we have

$$\begin{aligned}
Pr_{Tr(D,m)}(i:pf=v) &= M_{Tr(D,m)}(i:pf=v) \\
&= M_{D_m}(i:pf=v) \qquad (7.16)
\end{aligned}$$

for any pf constant pf and $v\in Dom(pf)$ and any $i\in\{0,\dots,m-1\}$.

Secondly, from Theorem 21, it can be seen that for any (probabilistic) stable model I of $Tr(D,m)$, $(\langle I|_{i:\sigma^{fl}}, I|_{i:\sigma^{act}}, I|_{i+1:\sigma^{fl}}\rangle, I|_{i:\sigma^{pf}})$ is always a pf-transition: the contrary would imply that for some stable model I of $Tr(D,m)$, there does not exist any assignment $TC \cup A$ on pf constants and action constants such that $I = I_{TC\cup A}$, which contradicts Theorem 21. Under Assumption 2, this implies

$$Pr_{Tr(D,m)}(k+1 : s' \mid k : s, k : e, k : pf) = \begin{cases} 1 & \text{if } (\langle s, a, s' \rangle, pf) \text{ is a pf-transition} \\ 0 & \text{otherwise} \end{cases}$$

and thus

$$Pr_{Tr(D,m)}(k+1 : s' \mid k : s, k : e, k : pf) = Pr_{D_m}(1 : s' \mid 0 : s, 0 : e, 0 : pf) \tag{7.17}$$

for all assignments pf to σ^{pf}.

From (7.16) and (7.17), and by Proposition 14, we have

$$
\begin{aligned}
& Pr_{Tr(D,m)}(k+1 : s' \mid k : s, k : e) \\
= & \{\text{Law of Total Probability}\} \\
= & \sum_{pf \text{ is any value assignment to } \sigma^{pf}} Pr_{Tr(D,m)}(k+1 : s' \mid k : s, k : e, k : pf) \cdot Pr_{Tr(D,m)}(k : pf) \\
= & \sum_{pf \text{ is any value assignment to } \sigma^{pf}} Pr_{Tr(D,m)}(k+1 : s' \mid k : s, k : e, k : pf) \cdot \\
& (\prod_{c \in \sigma^{pf}} Pr_{Tr(D,m)}(k : c = pf(c))) \\
= & \{\text{Proposition 14 and (7.16)}\} \\
= & \sum_{pf \text{ is any value assignment to } \sigma^{pf}} Pr_{Tr(D,m)}(k+1 : s' \mid k : s, k : e, k : pf) \cdot \\
& (\prod_{c \in \sigma^{pf}} M_{D_m}(k : c = pf(c))) \\
= & \{\text{From (7.17)}\} \\
= & \sum_{pf \text{ is any value assignment to } \sigma^{pf}} Pr_{D_m}(1 : s' \mid 0 : s, 0 : e, 0 : pf) \cdot (\prod_{c \in \sigma^{pf}} M_{D_m}(0 : c = pf(c))) \\
= & Pr_{D_m}(1 : s' \mid 0 : s, 0 : e)
\end{aligned}
$$

□

Corollary 2 *For every $m \geq 1$, X_m is a residual (probabilistic) stable model of $Tr(D,m)$ iff $X^0,\dots,X^{m-1}$ are transitions of D and $0\!:\!s_0$ is a residual stable model of D_{init}. Furthermore,*

$$Pr_{Tr(D,m)}(X_m \mid 0\!:\!e_0,\dots,m-1\!:\!e_{m-1}) = p(X^0)\times\cdots\times p(X^m)\times Pr_{Tr(D,m)}(0\!:\!s_0).$$

Proof. By Theorem 21, an interpretation I is a (probabilistic) stable model of $Tr(D,m)$ iff $I^0,\dots,I^{m-1}$ are pf-transitions and $(I_{TC\cup A})|_{0:\sigma^{fl}} \cup (I_{TC\cup A})|_{\sigma^{initpf}}$ is a residual stable model of $D_{init} \cup PF_0(D)$. From the definition of transition and pf-transition, it follows that X_m is a residual (probabilistic) stable model of D_m iff $X^0,\dots,X^{m-1}$ are transitions of D and $0 : s_0$ is a residual stable model of D_{init}.

Furthermore, we have

$$\begin{aligned}
&Pr_{Tr(D,m)}(X_m \mid 0:e_0,\dots,0:e_{m-1})\\
=\ &Pr_{Tr(D,m)}(m:s_m \mid m-1:s_{m-1}, m-1:e_{m-1})\cdot\\
&\cdots Pr_{Tr(D,m)}(2:s_2 \mid 1:s_1, 1:e_1)\cdot\\
&Pr_{Tr(D,m)}(1:s_1 \mid 0:s_0, 0:e_0)\cup Pr_{Tr(D,m)}(0:s_0)
\end{aligned}$$

We have

$$\begin{aligned}
&Pr_{Tr(D,m)}(X_m \mid s_0, e_0,\dots,e_{m-1})\\
=\ &\{\text{By Theorem 22}\}\\
=\ &Pr_{D_m}(1:s_m \mid 0:s_{m-1}, 0:e_{m-1})\cdot\cdots\cdot Pr_{D_m}(1:s_2 \mid 0:s_1, 0:e_1)\cdot\\
&Pr_{D_m}(1:s_1 \mid 0:s_0, 0:e_0)\cdot Pr_{Tr(D,m)}(0:s_0)\\
=\ &Pr_{D_m}(1:s_1 \mid 0:s_0, 0:e_0)\cdot Pr_{D_m}(1:s_2 \mid 0:s_1, 0:e_1)\cdot\cdots\cdot\\
&Pr_{D_m}(1:s_m \mid 0:s_{m-1}, 0:e_{m-1})\cdot Pr_{Tr(D,m)}(0:s_0)\\
=\ &p(X^1)\times\cdots\times p(X^m)\times Pr_{Tr(D,m)}(0:s_0).
\end{aligned}$$

□

Chapter 8

DECISION-THEORETIC LP$^{\text{MLN}}$

Many problems in AI are about how to make decisions that maximize the agent's utility. In this section, we define an extension of LP$^{\text{MLN}}$ to address this type of decision problems. This extension, called DT-LP$^{\text{MLN}}$ ("Decision-Theretic LP$^{\text{MLN}}$"), associates a utlity measure to each probabilistic stable model, in addition to the probability measure as defined before. We define decision evaluation and decision optimization problems under DT-LP$^{\text{MLN}}$ framework, illustrating how decision problems involving probabilistic reasoning can be modeled in DT-LP$^{\text{MLN}}$. We will also present an algorithm for decision optimization problem, adapted from the well-known MAXWalkSAT algorithm for finding most probable truth assignments. In Chapter 9, we will show how this extension of LP$^{\text{MLN}}$ leads to an extension of action language $p\mathcal{BC}+$ that can model sequential decision problems.

8.1 Extending LP$^{\text{MLN}}$ for Decision Theory

We extend the syntax and semantics of LP$^{\text{MLN}}$ for DT-LP$^{\text{MLN}}$ by introducing atoms of the form

$$\texttt{utility}(u, \mathbf{t}) \tag{8.1}$$

where u is a real number, and $\mathbf{t}$ is an arbitrary list of terms. These atoms can only occur in the head of hard rules of the form

$$\alpha : \texttt{utility}(u, \mathbf{t}) \leftarrow \mathit{Body} \tag{8.2}$$

where *Body* is a list of literals. We call these rules *utility rules.*

The weight and the probability of an interpretation are defined the same as in LP$^{\text{MLN}}$. The *utility* of an interpretation I under Π is defined as

$$U_\Pi(I) = \sum_{\texttt{utility}(u,\mathbf{t}) \in I} u.$$

Given a proposition A, the *expected utility* of A is defined as

$$E[U_\Pi(A)] = \sum_{I \models A} U_\Pi(I) \times P_\Pi(I \mid A). \tag{8.3}$$

A DT-LP$^{\text{MLN}}$ program is a pair $\langle \Pi, Dec \rangle$ where Π is an LP$^{\text{MLN}}$ program with a propositional signature σ (including `utility` atoms) and Dec is a subset of σ consisting of *decision atoms*. We consider two reasoning tasks on DT-LP$^{\text{MLN}}$.

- **Evaluating a Decision.** Given a propositional formula e ("evidence") and a truth assignment dec of decision atoms Dec, represented as a conjunction of literals over atoms in Dec, compute the expected utility of decision dec in the presence of evidence e, i.e., compute

$$E[U_\Pi(dec \wedge e)] = \sum_{I \models dec \wedge e} U_\Pi(I) \times P_\Pi(I \mid dec \wedge e).$$

- **Finding a Decision with Maximum Expected Utility (MEU)**. Given a propositional formula e ("evidence"), find the truth assignment dec on Dec such that the expected utility of dec in the presence of e is maximized, i.e., compute

$$\underset{dec\ :\ dec \text{ is a truth assignment on } Dec}{\text{argmax}} E[U_\Pi(dec \wedge e)]. \tag{8.4}$$

8.2 MaxWalkSAT based MEU Approximiation

Algorithm 3 is an approximate algorithm based on MaxWalkSAT for solving the MEU problem. For any truth assignment X on a set σ of atoms and an atom $v \in \sigma$, we use $X\ |_v$ to denote the truth assignment on σ obtained from X by flipping the truth value of v.

8.3 Using DT-LP$^{\text{MLN}}$ to Solve Decision Problems

We use the following example to illustrate how DT-LP$^{\text{MLN}}$ can be used to solve decision problems.

Example 14. *Consider a directed graph G representing a social network: (i) each vertex $v \in V(G)$ represents a person; each edge (v_1, v_2) represents that v_1 influences v_2; (ii) each edge $e = (v_1, v_2)$ is associated with a probability p_e representing the probability of the influence; (iii) each vertex v is associated with a cost c_v, representing the cost of marketing the product to v; (iv) each person who buys the product yields a reward of r.*

The goal is to choose a subset U of vertices as marketing targets so as to maximize the expected profit. The problem can be represented as a DT-LP$^{\text{MLN}}$ *program* Π^{market} *as follows:*

$$\alpha : buy(v) \leftarrow marketTo(v).$$

$$\alpha : buy(v_2) \leftarrow buy(v_1), influence(v_1, v_2).$$

$$\alpha : utility(r, v) \leftarrow buy(v).$$

with the graph instance represented as follows:

- *for each edge $e = (v_1, v_2)$, we introduce a probabilistic fact $ln(\frac{p_e}{1-p_e}) : influence(v_1, v_2)$;*
- *for each vertex $v \in V(G)$, we introduce the following rule:*
  $\alpha : \texttt{utility}(-c_v, v) \leftarrow marketTo(v).$

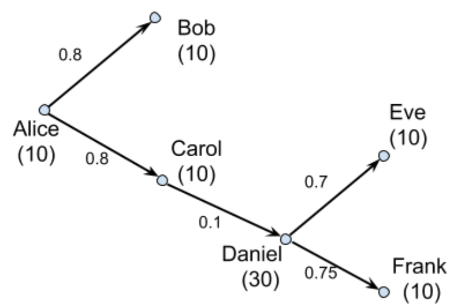
For simplicity, we assume that marketing to a person 100% *guarantees that the person buys the product. This assumption can be removed easily by changing the first rule to a soft rule.*

# People	# Edge	Solving Time ($DT-\mathrm{LP}^{\mathrm{MLN}}$; MC-ASP)	Solving Time (DT-ProbLog; Approximate)	Solving Time (DT-ProbLog; Exact)
10	28	9m45.677s	0.952s	1m3.725s
12	46	19m41.150s	17m45.320s	42m54.290s
14	58	29m56.269s	$>$ 4hr	$>$ 4hr
16	68	152m0.305s	$>$ 4hr	$>$ 4hr
18	84	226m5.618s	$>$ 4hr	$>$ 4hr
20	152	$>$ 4hr	$>$ 4hr	$>$ 4hr

Figure 8.1: Running Statistics of Algorithm 3 on Marketing Domain

The MEU solution of DT-LP$^{\mathrm{MLN}}$ *program* $(\Pi^{\mathrm{market}}, \{marketTo(v) \mid v \in V(G)\})$ *corresponds to the subset* U *of vertices that maximizes the expected profit.*

For example, consider the directed graph on the right, where each edge e *is labeled by* p_e *and each vertex* v *is labeled by* c_v*. Suppose the reward for each person buying the product is* 10*. There are* $2^6 = 64$ *different truth assignments on decision atoms, corresponding to* 64 *choices of marketing targets. The best decision is to market to* `Alice` *only, which yields the expected utility of* 17.96*.*



We implemented Algorithm 3 and report in Figure 8.1 its performance on the domain described in Example 14. We generate networks with $10, 12, \ldots, 20$ people and randomly generated edges, and use Algorithm 3 with MC-ASP as the underlying sampling methods for approximating expected utilities, DT-PROBLOG with exact mode and DT-PROBLOG with approximate mode resp., to find the optimal set of marketing targets. The graphs contain directed cycles. For Algorithm 3, 50 stable models are sampled to approximate each expected utility, p is set to be 0.5, m_t is set to be 10, and m_f is set to be 10. The experiments were performed on a machine powered by 4 Intel(R) Core(TM) i5-2400 CPU with OS Ubuntu 14.04.5 LTS and 8G memory.

8.4 Related Work

Nath and Domingos (2009) have introduced a decision-theoretic extension of Markov Logic, as a framework for relational decision theory based on Markov logic, where each clause is associated not only with a weight, but also a utility. Similar to DT-LP$^{\text{MLN}}$, each possible world is also associated with a utility, which is defined as the sum of utility of the clauses that the possible world satisfies. The MEU Problem is defined and computed with MAXWalkSAT base algorithm in a similar way. Despite the similarity in how the two frameworks are defined, the underlying stable model semantics of DT-LP$^{\text{MLN}}$ allows more compact representations of decision problems that require defeasible reasoning, causal reasoning, recursive definition, etc. For example, example 14 cannot be easily modeled with decision-theoretic MLN since it requires reasoning about transitive closure of relations.

Van den Broeck *et al.* (2010) introduce DT-PROBLOG, which is a decision-theoretic extension of ProbLog. DT-PROBLOG identify a set of atoms as decision atoms, which are special probabilistic facts whose probabilities are not assigned by the program but by a *strategy*. Utilities are assigned to arbitrary literals, from which the utility of a strategy is derived, as the expected total utility where the random variables are decision atoms, whose probabilities are defined by the strategy. The MEU problem is defined as finding a strategy (i.e., a probability distribution over truth assignment on decision facts) that maximizes the expected total utility.

Algorithm 3 MaxWalkSAT for Maximizing Expected Utility

Input:

1. (Π, A): A DT-LP$^{\text{MLN}}$ program;
2. E: a proposition in constraint form as the evidence;
3. m_t: the maximum number of tries;
4. m_f: the maximum number of flips;
5. p: probability of taking a random step.

Output: $soln$: a truth assignment on A

Process:

1. $soln \leftarrow null$;
2. $utility \leftarrow -\infty$;
3. For $i \leftarrow 1$ to m_t:
   (a) $X \leftarrow$ a random soft stable model of $\Pi \cup E$, found by LPMLN2ASP;
   (b) $soln' \leftarrow$ truth assignment of X on A;
   (c) $utility' \leftarrow E[U_\Pi(soln')]$;
   (d) For $j \leftarrow 1$ to m_f:
      i. If $Uniform(0,1) < p$:
         $v_f \leftarrow$ a randomly chosen decision atom;
         else:
         A. For each atom v in A:
            $DeltaCost(v) \leftarrow E[U_\Pi(soln' \wedge E)] - E[U_\Pi(soln' \mid_v \wedge E)]$;
         B. $v_f \leftarrow \underset{v: soln \mid_v \text{ is a partial stable model of } \Pi}{argmin} DeltaCost(v)$;
      ii. If $DeltaCost(v_f) < 0$:
         A. $soln' \leftarrow soln' \mid_{v_f}$;
         B. $utility' \leftarrow utility' - DeltaCost(v_f)$.
   (e) If $utility' > utility$:
      i. $utility \leftarrow utility'$;
      ii. $soln \leftarrow soln'$;

4. Return $soln$

Chapter 9

POLICY OPTIMIZATION AND RELATION TO (PARTIALLY OBSERVABLE) MARKOV DECISION PROCESS

In Chapter 7, we introduced the action language $p\mathcal{BC}+$, which can model action domains with probabilistic transitions. One important computational task in such domain is the planning task. Since actions may have stochastic effects, planning requires, rather than to find a sequence of actions that leads to a goal, to find an optimal policy, that states which actions to execute in each state to achieve the maximum expected utility.

In this section, we extend $p\mathcal{BC}+$ with the notion of utility, and define policy optimization problems in that language, in this way addressing planning problems in probabilistic action domains. The extension is defined as a high-level notation for DT-LP$^{\rm MLN}$. It turns out that the semantics of $p\mathcal{BC}+$ can also be directly defined in terms of Markov Decision Process (MDP), which in turn allows us to define MDP in a succinct and elaboration tolerant way. The result is theoretically interesting as it formally relates action languages to MDP despite their different origins, and furthermore justifies the semantics of the extended $p\mathcal{BC}+$ in terms of MDP. It is also computationally interesting because it allows for applying a number of algorithms developed for MDP to computing $p\mathcal{BC}+$ action descriptions. Based on this idea, we design the system PBCPLUS2MDP, which turns a $p\mathcal{BC}+$ action description into the input language of an MDP solver, and leverage MDP solving to find an optimal policy for the $p\mathcal{BC}+$ action description. Finally, we show that it is straightforward to extend $p\mathcal{BC}+$ to represent Partially Observable Markov Decision Process (POMDP).

9.1 $p\mathcal{BC}+$ with Utility

We extend $p\mathcal{BC}+$ by introducing the following expression called *utility law* that assigns a reward to transitions:

$$\textbf{reward } v \textbf{ if } F \textbf{ after } G \tag{9.1}$$

where v is a real number representing the reward, F is a formula that contains fluent constants only, and G is a formula that contains fluent constants and action constants only (no pf, no initpf constants). We extend the signature of $Tr(D, m)$ with a set of atoms of the form (8.1). We turn a utility law of the form (9.1) into the LP$^{\text{MLN}}$ rule

$$\alpha : \texttt{utility}(v, i+1, id) \leftarrow (i+1 : F) \wedge (i : G) \tag{9.2}$$

where id is a unique number assigned to this (ground) LP$^{\text{MLN}}$ rule and $i \in \{0, \dots, m-1\}$.

Given a nonnegative integer m denoting the maximum timestamp, a $p\mathcal{BC}+$ action description D with utility over multi-valued propositional signature σ is defined as a high-level representation of the DT-LP$^{\text{MLN}}$ program $(Tr(D, m), \sigma_m^{act})$.

We extend the definition of a probabilistic transition system as follows: A *probabilistic transition system* $T(D)$ represented by a probabilistic action description D is a labeled directed graph such that the vertices are the states of D, and the edges are obtained from the transitions of D: for every transition $\langle s, e, s' \rangle$ of D, an edge labeled $e : p, u$ goes from s to s', where $p = Pr_{D_1}(1 : s' \mid 0 : s \wedge 0 : e)$ and $u = E[U_{D_1}(0 : s \wedge 0 : e \wedge 1 : s')]$. The number p is called the *transition probability* of $\langle s, e, s' \rangle$, denoted by $p(s, e, s')$, and the number u is called the *transition reward* of $\langle s, e, s' \rangle$, denoted by $u(s, e, s')$.

Example 15. *The following action description D^{simple} describes a simple probabilistic action domain with two Boolean fluents P, Q, and two actions A and B. A causes*

P to be true with probability 0.8, and if P is true, then B causes Q to be true with probability 0.7. The agent receives the reward 10 if P and Q become true for the first time (after then, it remains in the state $\{P, Q\}$ as it is an absorbing state).

A **causes** P **if** Pf_1

B **causes** Q **if** $P \wedge Pf_2$

inertial P, Q

constraint $\neg(Q \wedge \sim P)$

caused $Pf_1 = \{\mathbf{t} : 0.8, \mathbf{f} : 0.2\}$

caused $Pf_2 = \{\mathbf{t} : 0.7, \mathbf{f} : 0.3\}$

reward 10 **if** $P \wedge Q$ **after** $\neg(P \wedge Q)$

caused $InitP = \{\mathbf{t} : 0.6, \mathbf{f} : 0.4\}$

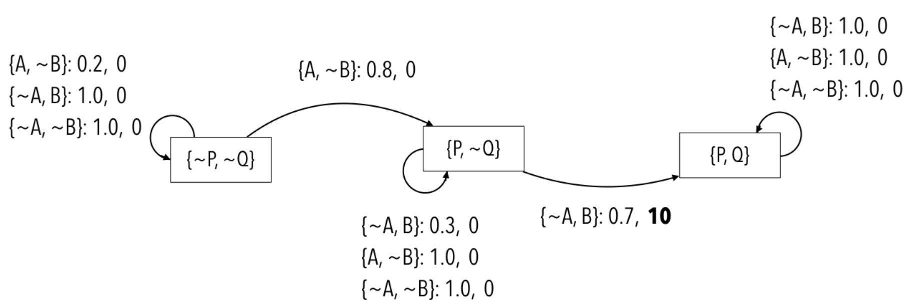
initially $P = x$ **if** $InitP = x$

caused $InitQ = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\}$

initially Q **if** $InitQ \wedge P$

initially $\sim Q$ **if** $\sim P$.

The transition system $T(D^{simple})$ is as follows:



9.2 Policy Optimization and Relation with Markov Decision Process

Given a $p\mathcal{BC}+$ action description D, we use $\mathbf{S}$ to denote the set of states, i.e, the set of interpretations I^{fl} of σ^{fl} such that $0 : I^{fl}$ is a residual (probabilistic) stable model of D_0. We use $\mathbf{A}$ to denote the set of interpretations I^{act} of σ^{act} such that $0 : I^{act}$ is a residual (probabilistic) stable model of D_1. Since we assume at most one action is executed each time step, each element in $\mathbf{A}$ makes either only one action or none to be true.

A *(non-stationary) policy* π (in $p\mathcal{BC}+$) is a function

$$\pi : \mathbf{S} \times \{0, \dots, m-1\} \mapsto \mathbf{A}$$

that maps a state and a time step to an action (including doing nothing). By $\langle s_0, s_1 \dots, s_m\rangle^t$ (each $s_i \in \mathbf{S}$) we denote the formula $0\!:\!s_0 \wedge 1\!:\!s_1 \dots \wedge m\!:\!s_m$, and by $\langle s_0, a_0, s_1 \dots, s_{m-1}, a_{m-1}, s_m\rangle^t$ (each $s_i \in \mathbf{S}$ and each $a_i \in \mathbf{A}$) the formula

$$0\!:\!s_0 \wedge 0\!:\!a_0 \wedge 1\!:\!s_1 \wedge \dots \wedge m-1\!:\!a_{m-1} \wedge m\!:\!s_m.$$

For any $i \in \{0, \dots, m\}$ and $s \in \mathbf{S}$, we write $i : s$ as an abbreviation of the formula $\bigwedge_{fl \in \sigma^{fl}} i : fl = s(fl)$; for any $i \in \{0, \dots, m-1\}$ and $a \in \mathbf{A}$, we write $i : a$ as an abbreviation of the formula $\bigwedge_{act \in \sigma^{act}} i\!:\!act = a(act)$.

We say a state s is *consistent* with D_{init} if there exists at least one probabilistic stable model I of D_{init} such that $I \models 0\!:\!s$. The *Policy Optimization* problem is to find a policy π that maximizes the expected utility starting from s_0, i.e., π with

$$\underset{\pi \text{ is a policy}}{\operatorname{argmax}} \; E[U_{Tr(\Pi,m)}(C_{\pi,m} \cup \langle s_0\rangle^t)]$$

where $C_{\pi,m}$ is the following formula representing policy π:

$$\bigwedge_{s \in \mathbf{S},\ \pi(s,i)=a,\ i \in \{0,\dots,m\}} i\!:\!s \rightarrow i\!:\!a \ .$$

We define the *total reward* of a history $\langle s_0, a_0, s_1, \dots, s_m\rangle$ *under action description* D as

$$R_D(\langle s_0, a_0, s_1, \dots, s_m\rangle) = E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, a_1, \dots, a_{m-1}, s_m\rangle^t)].$$

Although it is defined as an expectation, the following proposition tells us that any stable model X of $Tr(D, m)$ such that $X \models \langle s_0, a_0, s_1, \dots, s_m\rangle$ has the same utility, and consequently, the expected utility of $\langle s_0, a_0, s_1, \dots, s_m\rangle$ is the same as the utility of any single stable model that satisfies the history.

Proposition 15. *For any two stable models X_1, X_2 of $Tr(D, m)$ that satisfy a history $\langle s_0, a_0, s_1, a_1, \dots, a_{m-1}, s_m\rangle$, we have*

$$U_{Tr(D,m)}(X_1) \;=\; U_{Tr(D,m)}(X_2) \;=\; E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, a_1, \dots, a_{m-1}, s_m\rangle^t)].$$

It can be seen that the expected utility of π can be computed from the expected utility from all possible state sequences.

Proposition 16. *Given any initial state s_0 that is consistent with D_{init}, for any non-stationary policy π, we have*

$$E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0\rangle^t)] =$$
$$\sum_{\langle s_1,\dots,s_m\rangle : s_i \in \mathbf{S}} R_D(\langle s_0, \pi(s_0), s_1, \dots, \pi(s_{m-1}), s_m\rangle) \times P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m\rangle^t \mid \langle s_0\rangle^t \wedge C_{\pi,m}).$$

Definition 6. *For a p$\mathcal{BC}$+ action description D, let $M(D)$ be the MDP $\langle S, A, T, R\rangle$ where*

- *the state set S is $\mathbf{S}$;*
- *the action set A is $\mathbf{A}$;*
- *transition probability T is defined as $T(s, a, s') = P_{D_1}(1 : s' \mid 0 : s \wedge 0 : a)$;*
- *reward function R is defined as $R(s, a, s') = E[U_{D_1}(0 : s \wedge 0 : a \wedge 1 : s')]$.*

We show that the policy optimization problem for a p$\mathcal{BC}$+ action description D can be reduced to policy optimization problem for $M(D)$ for the finite horizon. The following theorem tells us that for any history following a non-stationary policy, its total reward and probability under D defined under the p$\mathcal{BC}$+ semantics coincide with those under the corresponding MDP M(D).

Theorem 23. *Given an initial state $s_0 \in \mathbf{S}$ that is consistent with D_{init}, for any non-stationary policy π and any finite state sequence $\langle s_0, s_1, \dots, s_{m-1}, s_m\rangle$ such that each s_i in $\mathbf{S}$ ($i \in \{0, \dots, m\}$), we have*

- $R_D(\langle s_0, \pi(s_0), s_1, \dots, \pi(s_{m-1}), s_m\rangle) = R_{M(D)}(\langle s_0, \pi(s_0,) \dots, \pi(s_{m-1}), s_m\rangle)$

- $P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m\rangle^t \mid \langle s_0\rangle^t \wedge C_{\pi,m}) = P_{M(D)}(\langle s_0, \pi(s_0,) \dots, \pi(s_{m-1}), s_m\rangle)$.

It follows that the policy optimization problem for $p\mathcal{BC}+$ action descriptions and the same problem for MDP with finite horizon coincide.

Theorem 24. *For any nonnegative integer m and an initial state $s_0 \in \mathbf{S}$ that is consistent with D_{init}, we have*

$$\underset{\pi \text{ is a non-stationary policy}}{\text{argmax}} E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0\rangle^t)] = \underset{\pi \text{ is a non-stationary policy}}{\text{argmax}} ER_{M(D)}(\pi, s_0).$$

Theorem 24 justifies using an implementation of DT-LP$^{\text{MLN}}$ to compute optimal policies of MDP $M(D)$ as well as using an MDP solver to compute optimal policies of the $p\mathcal{BC}+$ descriptions. Furthermore the theorems above allow us to check the properties of MDP $M(D)$ by using formal properties of LP$^{\text{MLN}}$, such as whether a certain state is reachable in a given number of steps.

9.3 $p\mathcal{BC}+$ as a High-Level Representation Language of MDP

An action description consists of causal laws in a human-readable form describing the action domain in a compact and high-level way, whereas it is non-trivial to describe an MDP instance directly from the domain description in English. The result in the previous section shows how to construct an MDP instance $M(D)$ for a $p\mathcal{BC}+$ action description D so that the solution to the policy optimization problem of D coincide with that of MDP $M(D)$. In that sense, $p\mathcal{BC}+$ can be viewed as a high-level representation language for MDP.

As its semantics is defined in terms of LP$^{\text{MLN}}$, $p\mathcal{BC}+$ inherits the nonmonotonicity of the stable model semantics to be able to compactly represent recursive definitions or transitive closure. The static laws in $p\mathcal{BC}+$ can prune out invalid states to ensure that only meaningful value combinations of fluents will be given to MDP as states, thus reducing the size of state space at the MDP level.

We illustrate the advantage of using $p\mathcal{BC}+$ action descriptions as high-level representations of MDP with an example.

Example 16. Robot and Blocks *There are two rooms* R1, R2, *and three blocks* B1, B2, B3 *that are originally located in* R1. *A robot can stack one block on top of another block if the two blocks are in the same room. The robot can also move a block to a different room, resulting in all blocks on top of it also moving if successful (with probability p). Each moving action has a cost of* 1. *What is the best way to move all blocks to* R2*?*

The example can be represented in $p\mathcal{BC}+$ as follows. x, x_1, x_2 range over B1, B2, B3*;* r, r_1, r_2 ranges over R1, R2. $\mathit{TopClear}(x)$, $\mathit{Above}(x_1, x_2)$, *and* $\mathit{GoalNotAchieved}$ *are Boolean statically determined fluent constants;* $\mathit{In}(x)$ *is a regular fluent constant with Domain* $\{\mathtt{R1}, \mathtt{R2}\}$, *and* $\mathit{OnTopOf}(x_1, x_2)$ *is a Boolean regular fluent constant.* $\mathit{MoveTo}(x, r)$ *and* $\mathit{StackOn}(x_1, x_2)$ *are action constants and* $\mathit{Pf\_Move}$ *is a Boolean pf constant. In this example, we make the goal state absorbing, i.e., when all the blocks are already in* R2*, then all actions have no effect.*

Moving block x to room r causes x to be in r with probability p:

$$\mathit{MoveTo}(x, r) \textbf{ causes } \mathit{In}(x) = r \textbf{ if } \mathit{Pf\_Move} \wedge \mathit{GoalNotAchieved}$$
$$\textbf{caused } \mathit{Pf\_Move} = \{\mathbf{t} : p, \mathbf{f} : 1 - p\}.$$

Successfully Moving a block x_1 to a room r_2 causes x_1 to be no longer underneath the block x_2 that x_1 was underneath in the previous step, if r_2 is different from where x_2 is:

$$\mathit{MoveTo}(x_1, r_2) \textbf{ causes } \sim \mathit{OnTopOf}(x_1, x_2)$$
$$\textbf{if } \mathit{Pf\_Move} \wedge \mathit{In}(x_1) = r_1 \wedge \mathit{OnTopOf}(x_1, x_2) \wedge \mathit{GoalNotAchieved} \qquad (r_1 \neq r_2).$$

Stacking a block x_1 on another block x_2 causes x_1 to be on top of x_2, if the top of x_2 is clear, and x_1 and x_2 are at the same location:

$$\begin{aligned} &\mathit{StackOn}(x_1, x_2) \textbf{ causes } \mathit{OnTopOf}(x_1, x_2) \\ &\qquad \textbf{if } \mathit{TopClear}(x_2) \wedge \mathit{At}(x_1) = r \wedge \mathit{At}(x_2) = r \wedge \mathit{GoalNotAchieved} \qquad (x_1 \neq x_2). \end{aligned}$$

Stacking a block x_1 on another block x_2 causes x_1 to be no longer on top of the block x where x_1 was originally on top of:

$$\begin{aligned} &\mathit{StackOn}(x_1, x_2) \textbf{ causes } \sim \mathit{OnTopOf}(x_1, x) \textbf{ if } \mathit{TopClear}(x_2) \wedge \mathit{At}(x_1) = r \wedge \mathit{At}(x_2) = r \wedge \\ &\qquad \mathit{OnTopOf}(x_1, x) \wedge \mathit{GoalNotAchieved} \qquad (x_2 \neq x, x_1 \neq x_2). \end{aligned}$$

Two different blocks cannot be on top of the same block, and a block cannot be on top of two different blocks:

$$\begin{aligned} &\textbf{constraint } \neg(\mathit{OnTopOf}(x_1, x) \wedge \mathit{OnTopOf}(x_2, x)) \qquad (x_1 \neq x_2) \\ &\textbf{constraint } \neg(\mathit{OnTopOf}(x, x_1) \wedge \mathit{OnTopOf}(x, x_2)) \qquad (x_1 \neq x_2). \end{aligned}$$

By default, the top of a block x is clear. It is not clear if there is another block x_1 that is on top of it:

$$\begin{aligned} &\textbf{default } \mathit{TopClear}(x) \\ &\textbf{caused } \sim \mathit{TopClear}(x) \textbf{ if } \mathit{OnTopOf}(x_1, x). \end{aligned}$$

The relation Above between two blocks is the transitive closure of the relation $OnTopOf$: A block x_1 is above another block x_2 if x_1 is on top of x_2, or there is another block x such that x_1 is above x and x is above x_2:

$$\begin{aligned} &\textbf{caused } \mathit{Above}(x_1, x_2) \textbf{ if } \mathit{OnTopOf}(x_1, x_2) \\ &\textbf{caused } \mathit{Above}(x_1, x_2) \textbf{ if } \mathit{Above}(x_1, x) \wedge \mathit{Above}(x, x_2). \end{aligned}$$

One block cannot be above itself; Two blocks cannot be above each other:

$$\textbf{caused } \bot \textbf{ if } \mathit{Above}(x_1, x_2) \wedge \mathit{Above}(x_2, x_1).$$

If a block x_1 is above another block x_2, then x_1 has the same location as x_2:

$$\textbf{caused } At(x_1) = r \textbf{ if } Above(x_1, x_2) \wedge At(x_2) = r. \tag{9.3}$$

Each moving action has a cost of 1:

$$\textbf{reward } -1 \textbf{ if } \top \textbf{ after } MoveTo(x, r).$$

Achieving the goal when the goal is not previously achieved yields a reward of 10:

$$\textbf{reward } 10 \textbf{ if } \sim GoalNotAchieved \textbf{ after } GoalNotAchieved.$$

The goal is not achieved if there exists a block x that is not at `R2`. *It is achieved otherwise:*

$$\textbf{caused } GoalNotAchieved \textbf{ if } At(x) = r \quad (r \neq \texttt{L2})$$
$$\textbf{default } \sim GoalNotAchieved.$$

$At(x)$ and $OnTopOf(x_1, x_2)$ are inertial:

$$\textbf{inertial } At(x), OnTopOf(x_1, x_2).$$

Finally, we add $a_1 \wedge a_2$ **causes** $\perp$ *for each distinct pair of ground action constants a_1 and a_2, to ensure that at most one action can occur each time step.*

It can be seen that stacking all blocks together and moving them at once would be the best strategy to move them to `L2`.

In Example 16, many value combinations of fluents do not lead to a valid state, such as

$$\{OnTopOf(\texttt{B1}, \texttt{B2}), OnTopOf(\texttt{B2}, \texttt{B1}), ...\}$$

where the two blocks `B1` and `B2` are on top of each other. Moreover, the fluents $TopClear(x)$ and $Above(x_1, x_2)$ are completely dependent on the value of the other fluents. There would be $2^{3+3\times3+3+3\times3} = 2^{24}$ states if we define a state as any value

# Blocks	# State	# Actions	LP$^{\text{MLN}}$ Solving Time	MDP Solving Time	Overall Solving Time
1	2	4	0.902s	0.0005	1.295s
2	8	9	0.958s	0.0014s	1.506s
3	44	16	1.634s	0.0017s	2.990s
4	304	25	12.256s	0.0347s	27.634s
5	2512	36	182.190s	2.502	10m23.929s
6	24064	49	> 1 hr	-	-

Figure 9.1: Running Statistics of PBCPLUS2MDP System

combination of fluents. On the other hand, the static laws in the above action descriptions reduce the number of states to only $(13 + 9) \times 2 = 44$.[1]

Furthermore, in this example, $Above(x, y)$ needs to be defined as a transitive closure of $OnTopOf(x, y)$, so that the effects of $StackOn(x_1, x_2)$ can be defined in terms of the (inferred) spatial relation of blocks. Also, the static law (9.3) defines an indirect effect of $MoveTo(x, r)$.

We implemented the prototype system PBCPLUS2MDP, which takes an action description D and time horizon m as input, and finds an optimal policy by constructing the corresponding MDP $M(D)$ and utilizing MDP policy optimization algorithms as blackbox. We use MDPTOOLBOX[2] as our underlying MDP solver. The current system uses LPMLN2ASP (see Chapter 5.1) for exact inference to find states, actions, transition probabilities, and transition rewards. The system is publicly available at `https://github.com/ywang485/pbcplus2mdp`, along with several examples.

We measure the scalability of our system PBCPLUS2MDP on Example 16. Figure 9.1 shows the running statistics of finding the optimal policy for different number of blocks. For all of the running instances, maximum time horizon is set to be 10, as in all of the instances, the smallest number of steps in a shortest possible action sequence achieving the goal is less than 10. The experiments are performed on a machine with 4 Intel(R) Core(TM) i5-2400 CPU with OS Ubuntu 14.04.5 LTS and 8 GB memory.

[1] This number can be verified by counting all possible configurations of 3 blocks with 2 locations.
[2] `https://pymdptoolbox.readthedocs.io`

As can be seen from the table, the running time increases exponentially as the number of blocks increases. This is not surprising since the size of the search space increases exponentially as the number of blocks increases. The bottleneck is the LP$^{\text{MLN}}$ inference system, as it needs to enumerate every stable model to generate the set of states, the set of actions, and transition probabilities and rewards. The time spent on MDP solving is negligible.

System PBCPLUS2MDP supports planning with infinite horizon. However, it should be noted that the semantics of an action description with infinite time horizon in terms of DT-LP$^{\text{MLN}}$ is not yet well established. In this case, the action description is only viewed as a high-level representation of an MDP.

9.4 Extending $p\mathcal{BC}+$ for representing POMDP

It is straightforward to extend $p\mathcal{BC}+$ so that it can be used as a high-level representation of Partially Observable Markov Decision Processes (POMDPs), which can be defined as a tuple

$$\langle S, A, T, R, \Omega, O\rangle$$

where

- S is a set of states;
- A is a set of actions;
- $T : S \times A \times S \rightarrow [0, 1]$ are transition probabilities;
- $R : S \times A \times S \rightarrow \mathbb{R}$ are rewards;
- Ω is a set of observations;
- $O : S \times A \times \Omega \rightarrow [0, 1]$ are observation probabilities.

We extend $p\mathcal{BC}+$ by introducing a new type of constants, called *observation constants*, and a new type of causal law called *observation dynamic law*. An observation dynamic law is of the form

$$\textbf{observe } F \textbf{ if } G \textbf{ after } H \tag{9.4}$$

where F contains observation constants only, G contains fluent constants only, and H contains action constants and/or pf constants only. Observation constants can only occur in observation dynamic laws. An observation dynamic law of the form (9.4) is translated into the following LP^{MLN} rule:

$$\alpha \ : \ (i+1\!:\!F) \leftarrow (i+1\!:\!G) \wedge (i\!:\!H)$$

For each observation constant *obs*, a special value `NA` ("Not Applicable") must be an element of $Dom(obs)$. For each observation constant in $obs \in \sigma^{obs}$ and $c \in Dom(obs)$, we include the following LP^{MLN} rule in D_m to indicate that the initial value of each observation constant is exogenous:

$$\alpha \ : \ \{0 : obs = c\}^{\text{ch}}.$$

and we include the following LP^{MLN} rule in D_m to indicate that by default, the value of *obs* is `NA`:

$$\alpha \ : \ \{i : obs = \texttt{NA}\}^{\text{ch}}.$$

for $i \in \{1, \dots, m\}$.

We use σ^{obs} to denote the set of observation constants. The signature σ_m of D_m is now extended with σ_m^{obs}.

For more flexible representations, we introduce special type of fluent constant called *rigid fluent constant*, which intuitively represent fluents whose values do not change over time steps. A *rigid static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{9.5}$$

where F and G contain rigid fluent constants only. Since the values of rigid fluent constants do not change over time steps, for any rigid fluent constant c and $i \in \{0, \dots, m\}$, we identify $i\!:\!c$ with c. A rigid static law (9.5) is translated into LP$^{\text{MLN}}$ rule

$$\alpha : F \leftarrow G$$

in D_m. We then extend pf constant declaration as

$$\textbf{caused } c = \{v_1 : p_1, \dots, v_n : p_n\} \textbf{ if } F \tag{9.6}$$

where c is a pf constant with domain $\{v_1, \dots, v_n\}$, $0 < p_i < 1$ for each $i \in \{1, \dots, n\}$, $\sum\limits_{i \in \{1,\dots,n\}} p_i = 1$ and F contains rigid fluent constants only. A pf constant declaration (9.6) is translated into LP$^{\text{MLN}}$ rules

$$ln(p_i) : (i : c) = v_j \leftarrow F \tag{9.7}$$

for $j \in \{0, \dots, m-1\}$. We define an expression of the form

$$\textbf{caused } c = \{v_1 : p_1, \dots, v_n : p_n\} \textbf{ unless } c$$

where c is a rigid fluent constant, as an abbreviation of

$$\textbf{caused } c = \{v_1 : p_1, \dots, v_n : p_n\} \textbf{ if } \sim c$$
$$\textbf{default} \sim c$$

We make the following assumption:

4. **Rigid Constants Take Same Value over All Stable Models**: for any rigid constant c, there exists $v \in Dom(c)$ such that $I \models c = v$ for all stable model I of D_m.

in addition to Assumption 1$\sim$3 listed in Section 7.2. Under this assumption, the body F in (9.7) evaluates to either $\mathbf{t}$ or $\mathbf{f}$ for all stable models of D_m, meaning that either

(9.7) can be removed from D_m, or F can be removed from the body of (9.7). We thus identify D_m as the LP$^{\rm MLN}$ program with such simplification performed, which is a k-coherent LP$^{\rm MLN}$ program.

A $p\mathcal{BC}+$ action description D defines a POMDP $M(D)$:

$$\langle S, A, P, R, \Omega, O\rangle$$

where

- state set S is $\mathbf{S}$;
- action set A is $\mathbf{A}$;
- transition probabilities P are defined as $P(s, a, s') = P_{D_1}(1 : s' \mid 0 : s, 0 : a)$;
- reward function R is defined as $R(s, a, s') = E[U_{D_1}(0 : s, 0 : a, 1 : s')]$;
- observation set Ω is the set of interpretations obs on σ^{obs} such that $0 : obs$ is a residual stable model of D_0;
- observation probabilities O are defined as $O(o, s, a) = P_{D_1}(1 : o \mid 1 : s, 0 : a)$ for all $s \in \mathbf{S}$ and $o \in \Omega$.

Example 17. Two Tigers Example *Consider a variation of the well-known tiger example with 2 tigers: there are three doors. There are two tigers behind two of the doors, and prize behind the other door. The agent does not know which object is behind which door. The agent can open any one of the three doors. The agent can also listen to get a better idea of where the tiger is. Listening yields the correct information about where each of the two tigers are with probability* 0.85*. This example can be represented with this extension of* $p\mathcal{BC}+$ *as follows:*

Notation: l, l_1, l_2, l_3 *range over* $\texttt{Left}$, $\texttt{Middle}$, $\texttt{Right}$, y *ranges over* $\texttt{Tiger1}$, $\texttt{Tiger2}$

Observation constant:	*Domains:*
$\mathit{TigerPositionObserved}(y)$	$\{\texttt{Left}, \texttt{Middle}, \texttt{Right}, \texttt{NA}\}$
Regular fluent constants:	*Domains:*
$\mathit{TigerPosition}(y)$	$\{\texttt{Left}, \texttt{Middle}, \texttt{Right}\}$
Action constants:	*Domains:*
Listen	*Boolean*
$\mathit{OpenDoor}(l)$	*Boolean*
Pf constants:	*Domains:*
Pf_Listen	*Boolean*
$\mathit{Pf\_FailedListen}(y)$	$\{\texttt{Left}, \texttt{Middle}, \texttt{Right}\}$

A reward of 10 *is obtained for opening the door with no tiger behind.*

$$\textbf{reward } 10 \textbf{ if } \mathit{TigerPosition}(\texttt{Tiger1}) = l_1 \wedge \mathit{TigerPosition}(\texttt{Tiger2}) = l_2 \textbf{ after } \mathit{OpenDoor}(l_3)$$
$$(l_1 \neq l_3, l_2 \neq l_3).$$

A penalty of 100 *is imposed for opening a door with tiger behind.*

$$\textbf{reward } -100 \textbf{ if } \mathit{TigerPosition}(y) = l \textbf{ after } \mathit{OpenDoor}(l).$$

Executing the action Listen has a small penalty of 1*.*

$$\textbf{reward } -1 \textbf{ if } \top \textbf{ after } \mathit{Listen}.$$

Two tigers cannot be at the same position.

$$\textbf{caused } \bot \textbf{ if } \mathit{TigerPosition}(\texttt{Tiger1}) = l \wedge \mathit{TigerPosition}(\texttt{Tiger2}) = l.$$

Successful listening reveals the positions of the two tigers.

$$\textbf{observe } \mathit{TigerPositionObserved}(y) = l \textbf{ if } \mathit{TigerPosition}(y) = l \textbf{ after } \mathit{Listen} \wedge \mathit{Pf\_Listen}.$$

Failed listening yield a random position for each tiger.

$$\textbf{caused } \mathit{Pf\_FailedListen}(y) = \{\texttt{Left}: \frac{1}{3}, \texttt{Middke}: \frac{1}{3}, \texttt{Right}: \frac{1}{3}\},$$

$$\textbf{observe } \mathit{TigerPositionObserved}(y) = l \textbf{ if } \top \textbf{ after } \mathit{Listen} \wedge \sim \mathit{Pf\_Listen} \wedge \mathit{Pf\_FailedListen}(y) = l.$$

The positions of tigers observe the commonsense law of inertia.

$$\textbf{inertial } \mathit{TigerPosition}(y).$$

The action Listen has a success rate of 0.85*.*

$$\textbf{caused } \mathit{Pf\_Listen} = \{\mathbf{t}: 0.85, \mathbf{f}: 0.15\}.$$

Various elaborations on this example can be easily achieved by changing a small part of the p$\mathcal{BC}$+ action description. For example, adding or removing tigers and doors requires simply adding or removing elements to/from the domains of the relevant constants, whereas such elaboration would require a complete reconstruction of transition/reward/observation matrices at POMDP level. This elaboration tolerance enables construction and solving of dynamic POMDPs.

We implemented the prototype system PBCPLUS2POMDP, which takes an action description D as input, and output the POMDP $M(D)$ in a standard format (`.pomdp`) that can be used as input to systems such as APPL[3]. The current system uses LPMLN2ASP (see Section 5.1) for exact inference to find states, actions, transition probabilities, observation probabilities and transition rewards. The system is publicly available at `https://github.com/ywang485/pbcplus2pomdp`, along with several examples.

We report the performance of the system on tiger example with increased number of tigers[4] in Figure 9.2.

[3]`http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`

[4]The number of doors is always number of tigers plus 1.

# Tigers	# States	# Actions	# Observations	LP^{MLN} Solving Time	POMDP Solving Time
1	2	4	3	2.294s	0.004s
2	6	5	13	2.889s	0.004s
3	24	6	73	18.687s	0.007s
4	120	7	501	11m55.735s	0.065s
5	720	8	4051	$>$ 1 hr	-

Figure 9.2: Running Statistics of PBCPLUS2POMDP System

9.5 $p\mathcal{BC}+$ as a Elaboration Tolerant Representation of POMDP

Consider the shopping request identification example from Zhang and Stone (2015): a delivery robot is responsible for buying an item i and deliver i to person p in room r. The robot needs to ask questions to figure out what i, p, r are. There are two types of questions that the robot can ask:

- Which-Questions: questions about what the item/person/room is, for example, "which item it is?"
- Confirmation-Questions: questions to confirm whether a(n) item/person/room is the requested one, for example, "is the requested item coffee?"

The speech recognition system is noisy, so the answer can be wrongly recognized. The robot can execute a `deliver` action, which has an item i', person p' and room r' as arguments. A reward is obtained with `deliver` action, determined by to what extent i', p' and r' matches i, p and r.

This example can be represented in $p\mathcal{BC}+$ as follows. For simplicity, we assume a small domain where $Item = \{Coffee, Coke, Cookies, Burger\}$, $Person = \{Alice, Bob, Carol\}$, $Room = \{R_1, R_2, R_3\}$.

Notation: i, i' range over $Item$, p, p' ranges over $Person$, r, r' ranges over $Room$, c ranges over $\{\texttt{Yes}, \texttt{No}\}$

Observation constant: Domains:

ObsItem	$Item \cup \{\texttt{NA}\}$
ObsPerson	$Person \cup \{\texttt{NA}\}$
ObsRoom	$Room \cup \{\texttt{NA}\}$
ObsYesOrNo	$\{\texttt{Yes}, \texttt{No}, \texttt{NA}\}$
Regular fluent constants:	Domains:
ItemRequested	$Item$
PersonRequested	$Person$
RoomRequested	$Room$
Terminated	Boolean
Action constants:	Domains:
AskWhichItem	Boolean
AskWhichPerson	Boolean
AskWhichRoom	Boolean
$Ask2ConfirmItem(i)$	Boolean
$Ask2ConfirmPerson(p)$	Boolean
$Ask2ConfirmRoom(r)$	Boolean
$Deliver(i, p, r)$	Boolean
Pf constants:	Domains:
$Pf\_AnswerWhichItem(i)$	$Item$
$Pf\_AnswerWhichPerson(p)$	$Person$
$Pf\_AnswerWhichRoom(r)$	$Room$
Pf_AnswerConsistentConfirm	$\{\texttt{Yes}, \texttt{No}\}$
Pf_AnswerInconsistentConfirm	$\{\texttt{Yes}, \texttt{No}\}$

The action *Deliver* causes the entering of the terminal state.

$$\textbf{caused } \mathit{Terminated} \textbf{ if } \top \textbf{after } \mathit{Deliver}(i, p, r).$$

Upon execution of *Deliver* action with a correct room, correct person and correct item will each yield a reward of 10; Execution of *Deliver* action with a wrong room will result in a penalty of 10.

$$\begin{aligned}
&\textbf{reward } 10 \textbf{ if } \mathit{RoomRequested} = r \wedge \mathit{PersonRequested} = p \\
&\quad \textbf{after } \mathit{Deliver}(i, p, r) \wedge \sim \mathit{Terminated}, \\
&\textbf{reward } 10 \textbf{ if } \mathit{RoomRequested} = r \wedge \mathit{ItemRequested} = i \\
&\quad \textbf{after } \mathit{Deliver}(i, p, r) \wedge \sim \mathit{Terminated} \\
&\textbf{reward } -10 \textbf{ if } \mathit{RoomRequested} = r \\
&\quad \textbf{after } \mathit{Deliver}(i, p, r') \wedge \sim \mathit{Terminated} \wedge r \neq r'.
\end{aligned}$$

Asking "which item" question when the actual item being requested is i returns a random item i' as observation, according to the probability distribution defined by pf constant *Pf_AnswerWhichItem*(i),

$$\begin{aligned}
&\textbf{observe } \mathit{ObsItem} = i' \textbf{ if } \mathit{ItemRequested} = i \wedge \sim \mathit{Terminated}\\
&\quad \textbf{after } \mathit{AskWhichItem} \wedge \mathit{Pf_AnswerWhichItem}(i) = i',\\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Coffee) = &(9.8)\\
&\quad \{Coffee : 0.7, Coke : 0.1, Cookies : 0.1, Burger : 0.1\},\\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Coke) = &(9.9)\\
&\quad \{Coffee : 0.1, Coke : 0.7, Cookies : 0.1, Burger : 0.1\},\\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Cookies) = &(9.10)\\
&\quad \{Coffee : 0.1, Coke : 0.1, Cookies : 0.7, Burger : 0.1\},\\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Burger) = &(9.11)\\
&\quad \{Coffee : 0.1, Coke : 0.1, Cookies : 0.1, Burger : 0.7\},
\end{aligned}$$

similar for asking “which person” and “which room” questions.

$$\begin{aligned}
&\textbf{observe } \mathit{ObsPerson} = p' \textbf{ if } \mathit{PersonRequested} = p \wedge \sim \mathit{Terminated}\\
&\quad \textbf{after } \mathit{AskPerson} \wedge \mathit{Pf_AnswerWhichPerson(p)} = p',\\
&\textbf{caused } \mathit{Pf_AnswerWhichPerson}(Alice) = \{Alice : 0.8, Bob : 0.1, Carol : 0.1\},\\
&\textbf{caused } \mathit{Pf_AnswerWhichPerson}(Bob) = \{Alice : 0.1, Bob : 0.8, Carol : 0.1\},\\
&\textbf{caused } \mathit{Pf_AnswerWhichPerson}(Carol) = \{Alice : 0.1, Bob : 0.1, Carol : 0.8\},\\
&\textbf{observe } \mathit{ObsRoom} = r' \textbf{ if } \mathit{PersonRequested} = r \wedge \sim \mathit{Terminated}\\
&\quad \textbf{after } \mathit{AskPerson} \wedge \mathit{Pf_AnswerWhichPerson(r)} = r',\\
&\textbf{caused } \mathit{Pf_AnswerWhichRoom}(R_1) = \{R_1 : 0.8, R_2 : 0.1, R_3 : 0.1\},\\
&\textbf{caused } \mathit{Pf_AnswerWhichRoom}(R_2) = \{R_1 : 0.1, R_2 : 0.8, R_3 : 0.1\},\\
&\textbf{caused } \mathit{Pf_AnswerWhichRoom}(R_3) = \{R_1 : 0.1, R_2 : 0.1, R_3 : 0.8\}.
\end{aligned}$$

Asking "is the item i" question when the actual item being requested is indeed i returns "yes" or "no" as observation, according to the probability distribution defined by pf constant *Pf_AnswerConsistentConfirm*,

$$\begin{aligned}
&\textbf{observe } \mathit{ObsYesOrNo} = c \textbf{ if } \mathit{ItemRequested} = i \wedge \sim \mathit{Terminated} \\
&\qquad \textbf{after } \mathit{Ask2ConfirmItem}(i) \wedge \mathit{Pf_AnswerConsistentConfirm} = c \\
&\textbf{caused } \mathit{Pf_AnswerConsistentConfirm} = \{\texttt{Yes} : 0.8, \texttt{No} : 0.2\}.
\end{aligned}$$

Asking "is the item i" question when the actual item being requested is a different item i' returns "yes" or "no" as observation, according to the probability distribution defined by pf constant *Pf_AnswerInconsistentConfirm*,

$$\begin{aligned}
&\textbf{observe } \mathit{ObsYesOrNo} = c \textbf{ if } \mathit{ItemRequested} = i \wedge \sim \mathit{Terminated} \\
&\qquad \textbf{after } \mathit{Ask2ConfirmItem}(i') \wedge \mathit{Pf_AnswerInconsistentConfirm} = c \wedge i \neq i', \\
&\textbf{caused } \mathit{Pf_AnswerInconsistentConfirm} = \{\texttt{Yes} : 0.2, \texttt{No} : 0.8\}.
\end{aligned}$$

The effects of asking "is the person p" or "is the room r" questions are defined similarly.

$$\begin{aligned}
&\textbf{observe } \mathit{ObsYesOrNo} = c \textbf{ if } \mathit{PersonRequested} = p \wedge \sim \mathit{Terminated} \\
&\qquad \textbf{after } \mathit{Ask2ConfirmPerson}(p) \wedge \mathit{Pf_AnswerConsistentConfirm} = c, \\
&\textbf{observe } \mathit{ObsYesOrNo} = c \textbf{ if } \mathit{PersonRequested} = p \wedge \sim \mathit{Terminated} \\
&\qquad \textbf{after } \mathit{Ask2ConfirmPerson}(p') \wedge \mathit{Pf_AnswerInconsistentConfirm} = c \wedge p \neq p', \\
&\textbf{observe } \mathit{ObsYesOrNo} = c \textbf{ if } \mathit{RoomRequested} = r \wedge \sim \mathit{Terminated} \\
&\qquad \textbf{after } \mathit{Ask2ConfirmRoom}(r) \wedge \mathit{Pf_AnswerConsistentConfirm} = c, \\
&\textbf{observe } \mathit{ObsYesOrNo} = c \textbf{ if } \mathit{RoomRequested} = r \wedge \sim \mathit{Terminated} \\
&\qquad \textbf{after } \mathit{Ask2ConfirmRoom}(r') \wedge \mathit{Pf_AnswerInconsistentConfirm} = c \wedge r \neq r'.
\end{aligned}$$

We illustrate that the above $p\mathcal{BC}+$ action description is elaboration tolerant through the following elaborations.

Elaboration 1: Unavailable items When an item is unavailable, we can simply remove that item from the domains of relevant constants. For example, suppose *Coke* is not available, then we simply need to replace (9.8) - (9.11) with

$$\textbf{caused } \mathit{Pf\_ItemAnswer}(Coffee) = \{Coffee : 0.78, \mathit{Cookies} : 0.11, \mathit{Burger} : 0.11\},$$

$$\textbf{caused } \mathit{Pf\_ItemAnswer}(\mathit{Cookies}) = \{Coffee : 0.11, \mathit{Cookies} : 0.78, \mathit{Burger} : 0.11\},$$

$$\textbf{caused } \mathit{Pf\_ItemAnswer}(\mathit{Burger}) = \{Coffee : 0.11, \mathit{Cookies} : 0.11, \mathit{Burger} : 0.78\}$$

Elaboration 2: Reflect personal preference in reward function We use a rigid fluent $\mathit{interchangeable}(p, i_1, i_2)$ with domain $[-10, 10]$ to represent how much two items i_1, i_2 are interchangeable according to the person p. For example

$$\textbf{caused } \mathit{interchangeable}(\mathit{Alice}, Coffee, \mathit{Coke}) = 5,$$

$$\textbf{caused } \mathit{interchangeable}(\mathit{Alice}, Coffee, \mathit{Cookies}) = 1,$$

$$\textbf{caused } \mathit{interchangeable}(\mathit{Alice}, Coffee, \mathit{Burger}) = -3$$

says that for Alice, bringing coke when she orders coffee still yields half of the reward, bring cookies when she orders coffee yields only 10% of the reward, and bringing burger when she orders coffee would yield a penalty which is 30% of the reward had the delivery been correct.

Then we add

$$\textbf{reward } x \textbf{ if } \mathit{ItemRequested} = i \wedge \mathit{interchangeable}(p, i, i') = x \wedge$$
$$\mathit{PersonRequested}(p) \wedge \mathit{RoomRequested}(r) \textbf{ after } \mathit{Deliver}(i', p, r)$$

Elaboration 3: Changing Perception Model The speech recognition system may have different accuracy depending on the environment. For example, when there is loud background noise, its accuracy could drop. In this case, we can simply plug in different probability distribution for the relevant pf constant, controlled by auxiliary constants indicating the situation. We introduce a rigid fluent called *Noise*, then we replace (9.8) - (9.11) with

$$\begin{aligned}
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Coffee) = \\
&\quad \{Coffee : 0.7, Coke : 0.1, Cookies : 0.1, Burger : 0.1\} \textbf{ unless } ab \\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Coke) = \\
&\quad \{Coffee : 0.1, Coke : 0.7, Cookies : 0.1, Burger : 0.1\} \textbf{ unless } ab \\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Cookies) = \\
&\quad \{Coffee : 0.1, Coke : 0.1, Cookies : 0.7, Burger : 0.1\} \textbf{ unless } ab \\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Burger) = \\
&\quad \{Coffee : 0.1, Coke : 0.1, Cookies : 0.1, Burger : 0.7\} \textbf{ unless } ab
\end{aligned}$$

to make them defeasible. We then define the probability distribution to override the original ones when there is loud background noise.

$$\begin{aligned}
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Coffee) = \\
&\quad \{Coffee : 0.4, Coke : 0.2, Cookies : 0.2, Burger : 0.2\} \textbf{ if } Noise, \\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Coke) = \\
&\quad \{Coffee : 0.2, Coke : 0.4, Cookies : 0.2, Burger : 0.2\} \textbf{ if } Noise, \\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Cookies) = \\
&\quad \{Coffee : 0.2, Coke : 0.2, Cookies : 0.4, Burger : 0.2\} \textbf{ if } Noise, \\
&\textbf{caused } \mathit{Pf_AnswerWhichItem}(Burger) = \\
&\quad \{Coffee : 0.2, Coke : 0.2, Cookies : 0.2, Burger : 0.4\} \textbf{ if } Noise.
\end{aligned}$$

We add

$$\textbf{caused } ab \textbf{ if } Noise.$$

to indicate that by default there is no background noise. When the robot agent detects that there is background noise, we add

$$\textbf{caused } Noise$$

to the action description to update the generated POMDP to incorporate the new speech recognition probabilities.

9.6 Related Work

There have been quite a few studies and attempts in defining factored representations of (PO)MDP, with feature-based state descriptions and more compact, human-readable action definitions. PPDDL (Younes and Littman (2004)) extends PDDL with constructs for describing probabilistic effects of actions and reward from state transitions. One limitation of PPDDL is the lack of static causal laws, which prohibits PPDDL from expressing recursive definitions or transitive closure. This may yield a large state space to explore as discussed in Section 9.3.

RDDL (Relational Dynamic Influence Diagram Language) (Sanner (2010)) improves the expressivity of PPDDL in modeling stochastic planning domains by allowing concurrent actions, continuous values of fluents, state constraints, etc. The semantics is defined in terms of lifted dynamic Bayes network extended with influence graph. A lifted planner can utilize the first-order representation and potentially achieve better performance. Still, indirect effects are hard to be represented in RDDL.

Zhang and Stone (2015) adopt ASP and P-Log (Baral *et al.* (2009)) to perform high-level symbolic reasoning, which, respectively produce a refined set of states and a refined probability distribution over states that are then fed to POMDP solvers for low-level planning. The refined sets of states and probability distribution over states take into account commonsense knowledge about the domain, and thus improves the quality of a plan and reduces computation needed at the POMDP level. Yang *et al.* (2018) adopts the (deterministic) action description language $\mathcal{BC}$ for high-level representations of the action domain, which defines high-level actions that can be treated as deterministic. Each action in the generated high-level plan is then mapped into more detailed low-level policies, which takes in stochastic effects of low-level actions into account. Similarly, Sridharan *et al.* (2015) introduce a framework with planning in a coarse-resolution transition model and a fine-resolution transition model. Action language $\mathcal{AL}_d$ is used for defining the two levels of transition models. The fine-resolution transition model is further turned into a POMDP for detailed planning with stochastic effects of actions and transition rewards. While a $p\mathcal{BC}+$ action description can fully capture all aspects of (PO)MDP including transition probabilities and rewards, the $\mathcal{AL}_d$ action description only provides states, actions and transitions with no quantitative information. Leonetti *et al.* (2016), on the other hand, use symbolic reasoners such as ASP to reduce the search space for reinforcement learning based planning methods by generating partial policies from planning results generated by the symbolic reasoner. The exploration of the low-level RL module is constrained by actions that satisfy the partial policy.

Another related work is Ferreira *et al.* (2017), which combines ASP and reinforcement learning by using action language $\mathcal{BC}+$ as a meta-level description of MDP. The $\mathcal{BC}+$ action descriptions define non-stationary MDPs in the sense that the states and actions can change with new situations occurring in the environment. The algorithm ASP(RL) proposed in this work iteratively calls ASP solver to obtain states and actions for the RL methods to learn transition probabilities and rewards, and update the

$\mathcal{BC}+$ action description with changes in the environment found by the RL methods, in this way finding optimal policy for a non-stationary MDP with the search space reduced by ASP. The work is similar to ours in that ASP-based high-level logical description is used to generate states and actions for MDP, but the difference is that we use an extension of $\mathcal{BC}+$ that expresses transition probabilities and rewards.

Although $p\mathcal{BC}+$ facilitates compact representations of (PO)MDPs, the planning systems presented here, PBCPLUS2(PO)MDP, translate $p\mathcal{BC}+$ action descriptions into ground (PO)MDPs, which could cause exponential blow-up in the size of the low-level (PO)MDP descriptions, as well as the sensitivity of (PO)MDP solving to ground size. It is worth noting that there are a few relational representations of (PO)MDPs that support lifted solving algorithms whose computation efficiency is not affected by ground size. For example, Sanner (2008) presents a first-order representation of (PO)MDPs based on situation calculus, where actions with stochastic effects are decomposed into a collection of deterministic actions corresponding to all possible outcomes of the original stochastic action, and every time the stochastic action is executed, "Nature" chooses one of its corresponding deterministic actions according to a probability distribution specified by a *case statement* - a mapping from first-order formulas to values. Case statements are also used for specifying reward functions and observation probabilities. Operations on case statements are defined for evaluating combinations of case statements at a first-order level. Lifted planning algorithms for (PO)MDPs represented in this way are then proposed, such as factored symbolic dynamic programming. A closely related first-order representation of (PO)MDPs that facilitates lifted solving algorithms is First-Order Decision Diagram (FODD, Wang *et al.* (2008)), which essentially uses labeled rooted directed acyclic graphs to represent case statements, used for representing stochastic effects of actions, reward functions and observation functions. Normal forms of FODDs, operations that turn arbitrary

FODDs into normal forms, as well as algorithms for combining FODDs, are introduced for efficient computation of case statements, and thus first-order (PO)MDP defined through case statements. Srivastava *et al.* (2014) proposed another first-order representation of (PO)MDP based on open-universe probability model BLOG (Milch *et al.* (2007)), called DT-BLOG. DT-BLOG can model open-universe (PO)MDPs where there can be uncertainty over the existence and identity of objects.

It would be interesting to investigate whether $p\mathcal{BC}+$ action descriptions can be turned into lifted representations of (PO)MDPs mentioned above, so as to utilize lifted (PO)MDP solving methods. We leave this as future work.

There are also modal logic based approach to formalizing action domains such as Moore (1984) and Morgenstern (1986), which consider very general settings where multi-agents and epistemic states of agents can be (at least partially) modeled.

9.7 Proofs of Proposition 15, Proposition 16, Theorem 23 and Theorem 24

It can be easily seen that Theorem 21, 22 and Corollary 2 still hold with the extension of $p\mathcal{BC}+$ with utility law.

We write $\langle a_0, a_1 \dots, a_{m-1}\rangle^t$ (each $a_i \in \mathbf{A}$) to denote the formula $0\!:\!a_0 \wedge 1\!:\!a_1 \dots \wedge m-1 : a_{m-1}$. The following lemma tells us that any action sequence has the same probability under $Tr(D, m)$.

Lemma 22. *For any* $p\mathcal{BC}+$ *action description* D *and any action sequence* $\langle a_0, a_1, \dots, a_{m-1}\rangle^t$, *we have*

$$P_{Tr(D,m)}(\langle a_0, a_1, \dots, a_{m-1}\rangle) = \frac{1}{(|\sigma^{act}| + 1)^m}.$$

Proof.

$$
\begin{aligned}
& P_{Tr(D,m)}(\langle a_0, a_1, \dots, a_{m-1}\rangle^t) \\
= & \sum_{\substack{I \models \langle a_0, a_1, \dots, a_{m-1}\rangle^t \\ I \text{ is a stable models of } Tr(D,m)}} P_{Tr(D,m)}(I) \\
= & \text{(In } Tr(D,m) \text{ every total choice leads to } (|\sigma^{act}|+1)^m \text{ stable models.} \\
& \text{By Proposition 2 in Lee and Wang (2018),)} \\
& \sum_{\substack{I \models \langle a_0, a_1, \dots, a_{m-1}\rangle^t \\ I \text{ is a stable models of } Tr(D,m)}} \frac{W_{Tr(D,m)}(I)}{(|\sigma^{act}|+1)^m} \\
= & \frac{\sum\limits_{tc \in TC_{Tr(D,m)}} \prod\limits_{c=v \in tc} M_\Pi(c=v)}{(|\sigma^{act}|+1)^m} \\
= & \text{(Derivations same as in the proof of Proposition 12)} \\
& \frac{1}{(|\sigma^{act}|+1)^m}
\end{aligned}
$$

□

The following lemma states that given any action sequence, the probabilities of all possible state sequences sum up to 1.

Lemma 23. *For any p$\mathcal{BC}$+ action description D and any action sequence $\langle a_0, a_1, \dots, a_{m-1}\rangle$, we have*

$$\sum_{s_0,\dots,s_m : s_i \in \mathbf{S}} P_{Tr(D,m)}(\langle s_0, \dots, s_m\rangle^t \mid \langle a_0, a_1, \dots, a_{m-1}\rangle^t) = 1.$$

Proof.

$$\sum_{s_0,\dots,s_m:s_i\in\mathbf{S}} P_{Tr(D,m)}(\langle s_0,\dots,s_m\rangle^t \mid \langle a_0,a_1,\dots,a_{m-1}\rangle^t)$$

$$= \text{(By Corollary 2)}$$

$$\sum_{s_0,\dots,s_m:s_i\in\mathbf{S}} \prod_{i\in\{0,\dots,m-1\}} p(s_i,a_i,s_{i+1})$$

$$= \sum_{s_0\in\mathbf{S}}(p(s_0)\cdot \sum_{s_1,\dots,s_m:s_i\in\mathbf{S}} \prod_{i\in\{1,\dots,m-1\}} p(s_i,a_i,s_{i+1}))$$

$$= \sum_{s_0\in\mathbf{S}}(p(s_0)\cdot \sum_{s_1\in\mathbf{S}}(p(s_0,a_0,s_1)\cdot \sum_{s_2,\dots,s_m:s_i\in\mathbf{S}} \prod_{i\in\{2,\dots,m-1\}} p(s_i,a_i,s_{i+1})))$$

$$= \sum_{s_0\in\mathbf{S}}(p(s_0)\cdot \sum_{s_1\in\mathbf{S}}(p(s_0,a_0,s_1)\cdot\dots\cdot \sum_{s_m\in\mathbf{S}} p(s_{m-1},a_i,s_m)\dots))$$

$$= 1.$$

□

The following proposition tells us that the probability of any state sequence conditioned on the constraint representation of a policy π coincide with the probability of the state sequence conditioned on the action sequence specified by π w.r.t. the state sequence.

Proposition 17. *For any p$\mathcal{BC}$+ action description D, state sequence $\langle s_0,s_1,\dots,s_m\rangle$, and a non-stationary policy π, we have*

$$P_{Tr(D,m)}(\langle s_0,s_1,\dots,s_m\rangle^t \mid C_{\pi,m}) =$$

$$P_{Tr(D,m)}(\langle s_0,s_1,\dots,s_m\rangle^t \mid 0\!:\!\pi(s_0,0)\wedge\dots\wedge m-1\!:\!\pi(s_m-1,m-1))$$

Proof.

$$
\begin{aligned}
& P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m\rangle^t \mid C_{\pi,m}) \\
= & \frac{P_{Tr(D,m)}(0\!:\!s_0 \wedge 1\!:\!s_1 \wedge \cdots \wedge m\!:\!s_m \wedge C_{\pi,m})}{P_{Tr(D,m)}(C_{\pi,m})} \\
= & \frac{P_{Tr(D,m)}(0\!:\!s_0 \wedge 0\!:\!\pi(s_0,0), 1\!:\!s_1 \wedge \cdots \wedge m-1\!:\!\pi(s_{m-1}, m-1) \wedge m\!:\!s_m)}{P_{Tr(D,m)}(C_{\pi,m})} \\
= & \frac{P_{Tr(D,m)}(0\!:\!\pi(s_0,0) \wedge 1\!:\!s_1 \wedge \cdots \wedge m-1\!:\!\pi(s_{m-1}, m-1) \wedge m\!:\!s_m \mid 0\!:\!s_0) \cdot P_{Tr(D,m)}(0\!:\!s_0)}{\sum\limits_{s'_0,\dots,s'_m : s'_i \in \mathbf{S}} P_{Tr(D,m)}(0\!:\!s'_0 \wedge 0\!:\!\pi(s'_0,0), 1\!:\!s'_1 \wedge \cdots \wedge m-1\!:\!\pi(s'_{m-1}, m-1) \wedge m\!:\!s'_m)}.
\end{aligned}
$$

We use $k(s_0, \dots, s_m)$ as an abbreviation of

$$P_{Tr(D,m)}(0\!:\!\pi(s_0,0) \wedge \cdots \wedge m-1\!:\!\pi(s_{m-1}, m-1)).$$

We have

$$
\begin{aligned}
& P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m\rangle^t \mid C_{\pi,m}) \\
= & \frac{P_{Tr(D,m)}(1\!:\!s_1 \wedge \cdots \wedge m\!:\!s_m \mid 0\!:\!s_0 \wedge 0\!:\!\pi(s_0,0) \wedge m-1\!:\!\pi(s_{m-1}, m-1)) \cdot P_{Tr(D,m)}(0\!:\!s_0) \cdot k(s_0, \dots, s_m)}{\sum\limits_{s'_0,\dots,s'_m : s'_i \in \mathbf{S}} P_{Tr(D,m)}(1\!:\!s'_1 \wedge \cdots \wedge m\!:\!s'_m \mid 0\!:\!s'_0 \wedge 0\!:\!\pi(s'_0,0) \wedge \cdots \wedge m-1\!:\!\pi(s'_{m-1}, m-1)) \cdot P_{Tr(D,m)}(0\!:\!s'_0) \cdot k(s'_0, \dots, s'_m)} \\
= & \text{(By Lemma 22, for any } s_0, \dots, s_m (s_i \in \mathbf{S}) \text{, we have } k(s_0, \dots, s_m) = (|\sigma^{act}| + 1)^m) \\
& \frac{P_{Tr(D,m)}(1\!:\!s_1 \wedge \cdots \wedge m\!:\!s_m \mid 0 : s_0 \wedge 0 : \pi(s_0,0) \wedge m-1 : \pi(s_{m-1}, m-1)) \cdot P_{Tr(D,m)}(0 : s_0) \cdot \frac{1}{(|\sigma^{act}|+1)^m}}{\sum\limits_{s'_0,\dots,s'_m : s'_i \in \mathbf{S}} P_{Tr(D,m)}(1\!:\!s'_1 \wedge \cdots \wedge m\!:\!s'_m \mid 0\!:\!s'_0 \wedge 0\!:\!\pi(s'_0,0) \wedge \cdots \wedge m-1\!:\!\pi(s'_{m-1}, m-1)) \cdot P_{Tr(D,m)}(0\!:\!s'_0) \cdot \frac{1}{(|\sigma^{act}|+1)^m}} \\
= & \frac{P_{Tr(D,m)}(1\!:\!s_1 \wedge \cdots \wedge m\!:\!s_m \mid 0\!:\!s_0 \wedge 0\!:\!\pi(s_0,0) \wedge m-1\!:\!\pi(s_{m-1}, m-1)) \cdot P_{Tr(D,m)}(0 : s_0)}{\sum\limits_{s'_0,\dots,s'_m : s'_i \in \mathbf{S}} P_{Tr(D,m)}(1\!:\!s'_1 \wedge \cdots \wedge m\!:\!s'_m \mid 0\!:\!s'_0 \wedge 0\!:\!\pi(s'_0,0) \wedge \cdots \wedge m-1\!:\!\pi(s'_{m-1}, m-1)) \cdot P_{Tr(D,m)}(0\!:\!s'_0)} \\
= & \text{(By Lemma 23, the denominator equals 1)} \\
& P_{Tr(D,m)}(1\!:\!s_1 \wedge \cdots \wedge m\!:\!s_m \mid 0\!:\!s_0 \wedge 0\!:\!\pi(s_0,0) \wedge m-1\!:\!\pi(s_{m-1}, m-1)) \cdot P_{Tr(D,m)}(0\!:\!s_0) \\
= & P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m\rangle^t \mid \langle \pi(s_0,0), \dots, \pi(s_m - 1, m-1)\rangle^t)
\end{aligned}
$$

□

The following proposition tells us that the expected utility of an action and state sequence can be computed by summing up the expected utility from each transition.

Proposition 18. *For any p$\mathcal{BC}$+ action description D and a history $\langle s_0, a_0, s_1, \dots, a_{m-1}, s_m\rangle$, such that there exists at least one stable model of $Tr(D,m)$ that satisfies $\langle s_0, a_0, s_1, \dots, a_{m-1}, s_m\rangle$, we have*

$$E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, \dots, s_{m-1}, a_{m-1}, s_m\rangle^t)] = \sum_{i \in \{0,\dots,m-1\}} u(s_i, a_i, s_{i+1}).$$

Proof. Let X be any stable model of $Tr(D,m)$ that satisfies $\langle s_0, a_0, s_1, \ldots, s_{m-1}, s_m\rangle^t$. By Proposition 15, we have

$$\begin{aligned}
& E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, \ldots, s_{m-1}, s_m\rangle^t)] \\
= \; & U_{Tr(D,m)}(X) \\
= \; & \sum_{i\in\{0,\ldots,m-1\}} \Big(\sum_{\substack{utility(v,i,\mathbf{x})\leftarrow(i+1:F)\wedge(i:G)\in Tr(D,m) \\ X \text{ satisfies } (i+1\,:\,F)\wedge(i\,:\,G)}} v \Big) \\
= \; & \sum_{i\in\{0,\ldots,m-1\}} \Big(\sum_{\substack{utility(v,0,\mathbf{x})\leftarrow(1:F)\wedge(0:G)\in Tr(D,m) \\ 0:X^i \text{ satisfies } (1\,:\,F)\wedge(0\,:\,G)}} v \Big) \\
= \; & \sum_{i\in\{0,\ldots,m-1\}} U_{Tr(D,1)}(0\!:\!X^i) \\
= \; & (\text{ By Proposition 15}) \\
& \sum_{i\in\{0,\ldots,m-1\}} E[U_{Tr(D,1)}(0:s_i, 0:a_i, 1:s_{i+1})] \\
= \; & \sum_{i\in\{0,\ldots,m-1\}} u(s_i, a_i, s_{i+1}).
\end{aligned}$$

□

The following proposition tells us that, for any states and actions sequence, any stable model of $Tr(D,m)$ that satisfies the sequence has the same utility. Consequently, the expected utility of the sequence can be computed by looking at any single stable model that satisfies the sequence.

Proposition 15 *For any two stable models X_1, X_2 of $Tr(D,m)$ that satisfy a particular states and actions sequence $\langle s_0, a_0, s_1, a_1, \ldots, a_{m-1}, s_m\rangle$, we have*

$$U_{Tr(D,m)}(X_1) \;=\; U_{Tr(D,m)}(X_2) \;=\; E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, a_1, \ldots, a_{m-1}, s_m\rangle^t)].$$

Proof. Since both X_1 and X_2 both satisfy $\langle s_0, a_0, s_1, a_1, \ldots, a_{m-1}, s_m\rangle^t$, X_1 and X_2 agree on truth assignment on $\sigma_m^{act} \cup \sigma_m^{fl}$. Notice that atom of the form $\mathtt{utility}(v, \mathbf{t})$

in $Tr(D,m)$ occurs only of the form (9.2), and only atom in $\sigma_m^{act} \cup \sigma_m^{fl}$ occurs in the body of rules of the form (9.2).

- Suppose an atom $\texttt{utility}(v, \mathbf{t})$ is in X_1. Then the body B of at least one rule of the form (9.2) with $\texttt{utility}(v, \mathbf{t})$ in its head in $Tr(D,m)$ is satisfied by X_1. B must be satisfied by X_2 as well, and thus $\texttt{utility}(v, \mathbf{t})$ is in X_2 as well.

- Suppose an atom $\texttt{utility}(v, \mathbf{t})$, is not in X_1. Then, assume, to the contrary, that $\texttt{utility}(v, \mathbf{t})$ is in X_2, then by the same reasoning process above in the first bullet, $\texttt{utility}(v, \mathbf{t})$ should be in X_1 as well, which is a contradiction. So $\texttt{utility}(v, \mathbf{t})$ is also not in X_2.

So X_1 and X_2 agree on truth assignment on all atoms of the form $\texttt{utility}(v, \mathbf{t})$, and consequently we have $U_{Tr(D,m)}(X_1) = U_{Tr(D,m)}(X_2)$, as well as

$$
\begin{aligned}
& E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, a_1, \dots, a_{m-1}, s_m \rangle^t)] \\
= & \sum_{I \models \langle s_0, a_0, s_1, \dots, a_{m-1}, s_m \rangle^t} P_{Tr(D,m)}(I \mid \langle s_0, a_0, s_1, \dots, a_{m-1}, s_m \rangle^t) \cdot U_{Tr(D,m)}(I) \\
= & \ U_{Tr(D,m)}(X_1) \cdot \sum_{I \models \langle s_0, a_0, s_1, \dots, a_{m-1}, s_m \rangle^t} P_{Tr(D,m)}(I \mid \langle s_0, a_0, s_1, \dots, a_{m-1}, s_m \rangle^t) \\
= & \ (\text{The second term equals 1}) \\
& U_{Tr(D,m)}(X_1).
\end{aligned}
$$

□

Proposition 16 *Given any initial state s_0 that is consistent with D_{init}, for any policy π, we have*

$$
\begin{aligned}
& E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0 \rangle^t)] = \\
& \sum_{\langle s_1, \dots, s_m \rangle : s_i \in \mathbf{S}} R_D(\langle s_0, \pi(s_0), s_1, \dots, \pi(s_{m-1}), s_m \rangle) \times P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m \rangle^t \mid \langle s_0 \rangle^t \wedge C_{\pi,m}).
\end{aligned}
$$

Proof. We have

$$
\begin{aligned}
& E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0 \rangle^t)] \\
= & \sum_{I \vDash 0:s_0 \wedge C_{\pi,m}} P_{Tr(D,m)}(I \mid 0\!:\!s_0 \wedge C_{\pi,m}) \cdot U_{Tr(D,m)}(I) \\
= & \sum_{\substack{I \vDash 0:s_0 \wedge C_{\pi,m} \\ I \text{ is a stable model of } Tr(D,m)}} P_{Tr(D,m)}(I \mid 0\!:\!s_0 \wedge C_{\pi,m}) \cdot U_{Tr(D,m)}(I) \\
= & \text{(We partition stable models } I \text{ according to their truth assignment on } \sigma_m^{fl}) \\
& \sum_{\langle s_1,\dots,s_m \rangle : s_i \in \mathbf{S}} \; \sum_{\substack{I \vDash \langle 0:s_0, 1:s_1, \dots, m:s_m \rangle^t \wedge C_{\pi,m} \\ I \text{ is a stable model of } Tr(D,m)}} P_{Tr(D,m)}(I \mid 0\!:\!s_0 \wedge C_{\pi,m}) \cdot U_{Tr(D,m)}(I) \\
= & \text{(Since } I \vDash \langle s_0, s_1, \dots, s_m \rangle^t \wedge C_{\pi,m} \text{ implies } I \vDash \langle s_0, \pi(s_0,0), s_1, \dots, s_m \rangle^t \text{, by Proposition 15 we have)} \\
& \sum_{\langle s_1,\dots,s_m \rangle : s_i \in \mathbf{S}} \; \sum_{\substack{I \vDash \langle s_0, s_1, \dots, s_m \rangle^t \wedge C_{\pi,m} \\ I \text{ is a stable model of } Tr(D,m)}} P_{Tr(D,m)}(I \mid 0\!:\!s_0 \wedge C_{\pi,m}) \cdot E[U_{Tr(D,m)}(\langle s_0, \pi(s_0,0), s_1, \dots, s_m \rangle^t)] \\
= & \sum_{\langle s_1,\dots,s_m \rangle : s_i \in \mathbf{S}} Pr_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m \rangle^t \mid 0\!:\!s_0 \wedge C_{\pi,m}) \cdot E[U_{Tr(D,m)}(\langle s_0, \pi(s_0,0), s_1, \dots, s_m \rangle^t)] \\
= & \sum_{\langle s_1,\dots,s_m \rangle : s_i \in \mathbf{S}} Pr_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m \rangle^t \mid 0\!:\!s_0 \wedge C_{\pi,m}) \cdot E[U_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m \rangle^t \wedge C_{\pi,m})] \\
= & \sum_{\langle s_1,\dots,s_m \rangle : s_i \in \mathbf{S}} R_D(\langle s_0, \pi(s_0), s_1, \dots, \pi(s_{m-1}), s_m \rangle) \times P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m \rangle^t \mid \langle s_0 \rangle^t \wedge C_{\pi,m}).
\end{aligned}
$$

□

Theorem 23 *Given an initial state $s_0 \in \mathbf{S}$ that is consistent with D_{init}, for any policy π and any finite state sequence $\langle s_0, s_1, \dots, s_{m-1}, s_m \rangle$ such that each s_i in $\mathbf{S}$ ($i \in \{0, \dots, m\}$), we have*

- $R_D(\langle s_0, \pi(s_0), s_1, \dots, \pi(s_{m-1}), s_m \rangle) = R_{M(D)}(\langle s_0, \pi(s_0,) \dots, \pi(s_{m-1}), s_m \rangle)$
- $P_{Tr(D,m)}(\langle s_0, s_1, \dots, s_m \rangle^t \mid \langle s_0 \rangle^t \wedge C_{\pi,m}) = P_{M(D)}(\langle s_0, \pi(s_0,) \dots, \pi(s_{m-1}), s_m \rangle)$.

Proof. We have

$$\begin{aligned}
& R_D(\langle s_0, \pi(s_0), s_1, \ldots, \pi(s_{m-1}), s_m\rangle) \\
= \; & E[U_{Tr(D,m)}(\langle s_0, s_1, \ldots, s_m\rangle^t \wedge C_{\pi,m})] \\
= \; & \text{(By Proposition 18)} \\
& \sum_{i \in \{0,\ldots,m-1\}} u(s_i, \pi(s_i, i), s_{i+1}) \\
= \; & \sum_{i \in \{0,\ldots,m-1\}} R(s_i, \pi(s_i, i), s_{i+1}) \\
= \; & R_{M(D)}(\langle s_0, \pi(s_0,) \ldots, \pi(s_{m-1}), s_m\rangle)
\end{aligned}$$

and

$$\begin{aligned}
& P_{Tr(D,m)}(\langle s_0, s_1, \ldots, s_m\rangle^t \mid \langle s_0\rangle^t \wedge C_{\pi,m}) \\
= \; & \text{(By Proposition 17)} \\
& Pr_{Tr(D,m)}(\langle s_0, s_1, \ldots, s_m\rangle \mid s_0 \wedge 0{:}\pi(s_0, 0) \wedge \cdots \wedge m-1{:}\pi(s_{m-1}, m-1)) \\
= \; & \text{(By Corollary 2)} \\
& \prod_{i \in \{0,\ldots,m-1\}} p(\langle s_i, \pi(s_i, i), s_{i+1}\rangle) \\
= \; & P_{M(D)}(\langle s_0, \pi(s_0,) \ldots, \pi(s_{m-1}), s_m\rangle)
\end{aligned}$$

□

Theorem 24 *For any nonnegative integer m and an initial state $s_0 \in \mathbf{S}$ that is consistent with D_{init}, we have*

$$\underset{\pi \text{ is a policy}}{\operatorname{argmax}} \; E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0\rangle^t)] = \underset{\pi}{\operatorname{argmax}} \; ER_{M(D)}(\pi, s_0).$$

Proof. We show that for any non-stationary policy π,

$$E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0\rangle^t)] = ER_{M(D)}(\pi, s_0).$$

We have

$$E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0\rangle^t)]$$

$=$ (By Proposition 16)

$$\sum_{\langle s_1,\ldots,s_m\rangle : s_i \in \mathbf{S}} R_D(\langle s_0, \pi(s_0), s_1, \ldots, \pi(s_{m-1}), s_m\rangle) \times P_{Tr(D,m)}(\langle s_0, s_1, \ldots, s_m\rangle^t \mid \langle s_0\rangle^t \wedge C_{\pi,m}).$$

$=$ (By Theorem 23)

$$\sum_{\langle s_1,\ldots,s_m\rangle : s_i \in \mathbf{S}} R_{M(D)}(\langle s_0, \pi(s_0, 0) \ldots, \pi(s_{m-1}, m-1), s_m\rangle) \cdot$$

$$\mathbb{P}_{M(D)}(\langle s_0, \pi(s_0, 0) \ldots, \pi(s_{m-1}, m-1), s_m\rangle)$$

$= ER_{M(D)}(\pi, s_0).$

$\square$

Chapter 10

CONCLUSION

Building intelligent agents in many real-world domains requires both complex reasoning (such as defeasible reasoning, causal reasoning, diagnostic reasoning etc.) and probabilistic inference, as well as the ability to utilize knowledge from human exports and learn from data statistically. Answer Set Programming has well addressed the problem of complex reasoning with the nonmonotonicity of the stable model semantics and allows easy representation of human knowledge. However, the crispy nature of the semantics brought difficulties in probabilistic reasoning and utilizing statistical information from data.

This research proposes the language $\mathrm{LP}^{\mathrm{MLN}}$, which introduces weighted logic rules under the stable model semantics, to extend ASP for probabilistic reasoning and statistical learning. $\mathrm{LP}^{\mathrm{MLN}}$ is a novel combination of ASP and the SRL formalism Markov Logic. It provides versatile methods to overcome the deterministic nature of the stable model semantics, such as resolving inconsistencies in answer set programs, ranking stable models, associating probability to stable models, and applying statistical inference to computing weighted stable models. It brings the learning aspect from Markov Logic to the answer set programming setting. $\mathrm{LP}^{\mathrm{MLN}}$ is also related to many other formalisms in SRL. As a middle-ground language, it helps understand the relation between those languages, and an $\mathrm{LP}^{\mathrm{MLN}}$ system can be used to compute those languages.

The prototype $\mathrm{LP}^{\mathrm{MLN}}$ systems presented in this dissertation, which automated $\mathrm{LP}^{\mathrm{MLN}}$ inference and weight learning, serve as a proof-of-concept of our reasoning and learning framework. Thanks to the sophisticated answer set optimization algorithms implemented in CLINGO, LPMLN2ASP system has a decent performance on MAP inference. However, as can be seen from a few experiments, the current system is not very scalable on marginal/conditional probability computation as well as weight learning. We expect incorporating advancements in both ASP and SRL will result in a more mature system that achieves better performance. For example, knowledge compilation techniques that turn logic programs into canonical representations, such as Sentential Decision Diagrams (SDD)(Vlasselaer *et al.* (2014)), where inference can be performed significantly faster. This was evidenced by similar approach on other probabilistic programming languages (see Section 5.5).

Here, we summarize the major contributions of the thesis and some directions for future work.

10.1 Summary of Contributions

We summarize the contributions of this thesis as follows:

- **We defined language** LP$^{\text{MLN}}$, and studied its theoretical properties.
- **We established the formal relationships between** LP$^{\text{MLN}}$ **and some other formalisms in KR and SRL**, including ASP with weak constraints, Markov Logic, ProbLog, Pearl's Probabilistic Causal Model, and P-log.
- **We developed** LP$^{\text{MLN}}$ **inference algorithms**, and implemented them as systems LPMLN2ASP and LPMLN2MLN. System LPMLN2ASP translates LP$^{\text{MLN}}$ programs into the input language of answer set solver CLINGO, and using weak constraints and stable model enumeration, it can compute most probable stable models as well as exact conditional and marginal probabilities. System LPMLN2MLN translates LP$^{\text{MLN}}$ programs into the input language of Markov Logic solvers, such as ALCHEMY, TUFFY, and ROCKIT, and allows for per-

forming approximate probabilistic inference on LP$^{\text{MLN}}$ programs.

- **We developed** LP$^{\text{MLN}}$ **weight learning algorithms** and implemented them as prototype system LPMLN-LEARN. We illustrated through examples that learning in LP$^{\text{MLN}}$ is in accordance with the stable model semantics, thereby it learns parameters for probabilistic extensions of knowledge-rich domains where answer set programming has shown to be useful but limited to the deterministic case, such as reachability analysis and reasoning about actions in dynamic domains.

- **We defined the action language** $p\mathcal{BC}+$ **as a high-level notion of** LP$^{\text{MLN}}$**, for modeling stochastic action domains**. We showed how probabilistic reasoning about transition systems, such as prediction, postdiction, and planning problems, as well as probabilistic diagnosis for dynamic domains, can be modeled in $p\mathcal{BC}+$ and computed using an implementation of LP$^{\text{MLN}}$.

- **We defined** DT-LP$^{\text{MLN}}$**, which is a decision theoretical extension of** LP$^{\text{MLN}}$. We defined reasoning tasks in DT-LP$^{\text{MLN}}$ and presented algorithms for the tasks.

- **We extended** $p\mathcal{BC}+$ **with the notion of utility, defined policy optimization problem under** $p\mathcal{BC}+$**, and formally related policy optimization problem under** $p\mathcal{BC}+$ **with that under Markov Decision Process**. The result showed that $p\mathcal{BC}+$ policy optimization problems can be computed with MDP solvers.

- **We implemented systems** PBCPLUS2(PO)MDP**, which turns** LP$^{\text{MLN}}$ **translations of** $p\mathcal{BC}+$ **action descriptions into (PO)MDP instances**. The systems allow for representing (PO)MDP in a succinct and elaboration tolerant way as well as leveraging an MDP solver to compute a $p\mathcal{BC}+$ action description.

10.2 Future Directions

A few interesting directions for future work include:

- **Develop more efficient inference/learning algorithms** incorporating advancements in both ASP and SRL, such as SDD based weighted model counting. As we mentioned before, this thesis focuses on exploring the expressivity of $\mathrm{LP}^{\mathrm{MLN}}$, and the current prototype systems are not yet very scalable on marginal/conditional probability computation as well as weight learning. For $\mathrm{LP}^{\mathrm{MLN}}$ to be applicable to real-world applications, dedicated research on more efficient inference/learning algorithms will be necessary.

- **Develop systems for inference in** DT-$\mathrm{LP}^{\mathrm{MLN}}$. In Chapter 8, we defined the inference tasks on a DT-$\mathrm{LP}^{\mathrm{MLN}}$ program and presented algorithms for them. An implementation of DT-$\mathrm{LP}^{\mathrm{MLN}}$ is yet to be developed.

- **Develop** $\mathrm{LP}^{\mathrm{MLN}}$ **structure learning algorithms that automatically construct** $\mathrm{LP}^{\mathrm{MLN}}$ **rules that fit the training data.** The problem of structure learning in $\mathrm{LP}^{\mathrm{MLN}}$ is about either generating an $\mathrm{LP}^{\mathrm{MLN}}$ program from scratch, or correcting a hand-crafted $\mathrm{LP}^{\mathrm{MLN}}$ program, so that the program best fits the training data given. This will facilitate a fully data-driven way of $\mathrm{LP}^{\mathrm{MLN}}$ modeling. We expect insights can be gained from Inductive Logic Programming (ILP) and MLN structure learning.

- **Develop a $p\mathcal{BC}+$ compiler that automates the translation from $p\mathcal{BC}+$ to** $\mathrm{LP}^{\mathrm{MLN}}$. Currently, the action language $p\mathcal{BC}+$ can be executable only through manual translation to $\mathrm{LP}^{\mathrm{MLN}}$. It is desirable to have a compiler that automates this translation, so that the user can directly write $p\mathcal{BC}+$ action descriptions and does not need to worry about the translation detail.

- **Conduct empirical study of** LP$^{\text{MLN}}$ **inference and learning on real-world applications.** LP$^{\text{MLN}}$ can be applied to problem domains that require both logical and probabilistic reasoning, both human knowledge and statistical information from data. Such domains include probabilistic extensions of combinatorial search problems, network modeling, diagnosis in stochastic transition systems, etc. Empirical study on these domains will be necessary to evaluate a LP$^{\text{MLN}}$ framework.

Babb, J. and J. Lee, “Action language $\mathcal{BC}+$”, Journal of Logic and Computation p. exv062, URL +http://dx.doi.org/10.1093/logcom/exv062 (2015).

Balai, E. and M. Gelfond, “On the relationship between P-log and LP$^{\text{MLN}}$”, in “Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)”, pp. 915–921 (2016).

Balduccini, M. and M. Gelfond, “Diagnostic reasoning with A-Prolog”, Theory and Practice of Logic Programming **3**, 425–461 (2003).

Baral, C., M. Gelfond and J. N. Rushton, “Probabilistic reasoning with answer sets”, Theory and Practice of Logic Programming **9**, 1, 57–144 (2009).

Baral, C., M. Gelfond and N. Rushton, “Probabilistic reasoning with answer sets”, in “Logic Programming and Nonmonotonic Reasoning”, edited by V. Lifschitz and I. Niemelä, pp. 21–33 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004).

Baral, C. and M. Hunsaker, “Using the probabilistic logic programming language p-log for causal and counterfactual reasoning and non-naive conditioning”, in “Proceedings of the 20th International Joint Conference on Artifical Intelligence”, IJCAI’07, pp. 243–249 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007), URL http://dl.acm.org/citation.cfm?id=1625275.1625313.

Baral, C., S. Mcilraith and T. Son, “Formulating diagnostic problem solving using an action language with narratives and sensing”, (2000).

Baral, C., N. Tran and L.-C. Tuan, “Reasoning about actions in a probabilistic setting”, in “Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)”, pp. 507–512 (2002).

Bellman, R., “A markovian decision process”, Indiana Univ. Math. J. **6**, 679–684 (1957).

Bellodi, E., E. Lamma, F. Riguzzi, V. S. Costa and R. Zese, “Lifted variable elimination for probabilistic logic programming”, Theory and Practice of Logic Programming **14**, 4-5, 681–695 (2014).

Bochman, A. and V. Lifschitz, “Pearl’s causality in a logical setting”, in “Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)”, (2015).

Buccafurri, F., N. Leone and P. Rullo, “Enhancing disjunctive datalog by constraints”, IEEE Transactions on Knowledge and Data Engineering **12**, 5, 845–860 (2000).

Calimeri, F., W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca and T. Schaub, “ASP-Core-2: Input language format”, ASP Standardization Working Group, Tech. Rep (2012).

D'Asaro, F. A., A. Bikakis, L. Dickens and R. Miller, "Foundations for a probabilistic event calculus", CoRR **abs/1703.06815**, URL `http://arxiv.org/abs/1703.06815` (2017).

De Raedt, L. and A. Kimmig, "Probabilistic (logic) programming concepts", Machine Learning **100**, 1, 5–47, URL `https://doi.org/10.1007/s10994-015-5494-z` (2015).

Denecker, M. and E. Ternovska, "Inductive situation calculus", Artificial Intelligence **171**, 5-6, 332–360 (2007).

Domingos, P. and D. Lowd, *Markov Logic: An Interface Layer for Artificial Intelligence*, Synthesis Lectures on Artificial Intelligence and Machine Learning (Morgan & Claypool Publishers, 2009).

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

Erdem, E. and V. Lifschitz, "Tight logic programs", Theory and Practice of Logic Programming **3**, 499–518 (2003).

Faber, W., N. Leone and G. Pfeifer, "Recursive aggregates in disjunctive logic programs: Semantics and complexity", in "Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)", (2004).

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

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

Ferraris, P., J. Lee and V. Lifschitz, "Stable models and circumscription", Artificial Intelligence **175**, 236–263 (2011).

Ferraris, P., J. Lee, V. Lifschitz and R. Palla, "Symmetric splitting in the general theory of stable models", in "Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)", pp. 797–803 (AAAI Press, 2009).

Ferreira, L. A., R. A. C. Bianchi, P. E. Santos and R. L. de Mantaras, "Answer set programming for non-stationary markov decision processes", Applied Intelligence **47**, 4, 993–1007, URL `https://doi.org/10.1007/s10489-017-0988-y` (2017).

Fierens, D., G. Van den Broeck, M. Bruynooghe and L. De Raedt, "Constraints for probabilistic logic programming", in "Proceedings of the NIPS Probabilistic Programming Workshop", pp. 1–4 (2012).

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

Gebser, M., R. Kaminski, B. Kaufmann and T. Schaub, "Multi-criteria optimization in answer set programming", in "LIPIcs-Leibniz International Proceedings in Informatics", vol. 11, pp. 1–10 (Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011).

Gebser, M., B. Kaufmann and T. Schaub, "Conflict-driven answer set solving: From theory to practice", Artificial Intelligence **187-188**, 52 – 89, URL `http://www.sciencedirect.com/science/article/pii/S0004370212000409` (2012).

Gebser, M., T. Schaub, S. Marius and S. Thiele, "XORRO: Near uniform sampling of answer sets by means of xor", Https://potassco.org/labs/2016/09/20/xorro.html (2016).

Giunchiglia, E., J. Lee, V. Lifschitz, N. McCain and H. Turner, "Nonmonotonic causal theories", Artificial Intelligence **153(1–2)**, 49–104 (2004).

Gomes, C. P., A. Sabharwal and B. Selman, "Near-uniform sampling of combinatorial spaces using xor constraints", in "Advances in Neural Information Processing Systems 19", edited by B. Schölkopf, J. C. Platt and T. Hoffman, pp. 481–488 (MIT Press, 2007), URL `http://papers.nips.cc/paper/3013-near-uniform-sampling-of-combinatorial-spaces-using-xor-constraints.pdf`.

Gutmann, B., *On Continuous Distributions and Parameter Estimation in Probabilistic Logic Programs*, Ph.D. thesis, KU Leuven (2011).

Gutmann, B., A. Kimmig, K. Kersting and L. De Raedt, "Parameter learning in probabilistic databases: A least squares approach", in "Joint European Conference on Machine Learning and Knowledge Discovery in Databases", pp. 473–488 (Springer, 2008).

Iwan, G., "History-based diagnosis templates in the framework of the situation calculus", AI Communications **15**, 1, 31–45 (2002).

Khot, T., S. Natarajan, K. Kersting and J. Shavlik, "Gradient-based boosting for statistical relational learning: the markov logic network and missing data cases", Machine Learning **100**, 1, 75–100, URL `https://doi.org/10.1007/s10994-015-5481-4` (2015).

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

Lee, J. and V. Lifschitz, "Loop formulas for disjunctive logic programs", in "Proceedings of International Conference on Logic Programming (ICLP)", pp. 451–465 (2003).

Lee, J. and Y. Meng, "First-order stable model semantics and first-order loop formulas", Journal of Artificial Inteligence Research (JAIR) **42**, 125–180 (2011).

Lee, J., Y. Meng and Y. Wang, “Markov logic style weighted rules under the stable model semantics”, **1433** (2015).

Lee, J., S. Talsania and Y. Wang, “Online appendix for the paper ”computing lpmln using asp and mln solvers””, (2017).

Lee, J. and Y. Wang, “Weighted rules under the stable model semantics”, in “Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)”, pp. 145–154 (2016).

Lee, J. and Y. Wang, “A probabilistic extension of action language BC+”, CoRR **abs/1805.00634**, URL `http://arxiv.org/abs/1805.00634` (2018).

Lee, J. and Z. Yang, “LPMLN, weak constraints, and P-log”, in “Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)”, pp. 1170–1177 (2017).

Leonetti, M., L. Iocchi and P. Stone, “A synthesis of automated planning and reinforcement learning for efficient, robust decision-making”, Artificial Intelligence **241**, 103 – 130 (2016).

Lifschitz, V., “What is answer set programming?”, in “Proceedings of the AAAI Conference on Artificial Intelligence”, pp. 1594–1597 (MIT Press, 2008).

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

Lifschitz, V. and H. Turner, “Representing transition systems by logic programs”, in “Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)”, pp. 92–106 (1999).

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

Mihalkova, L. and M. Richardson, “Speeding up inference in statistical relational learning by clustering similar query literals”, in “International Conference on Inductive Logic Programming”, pp. 110–122 (Springer, 2009).

Milch, B., B. Marthi, S. Russell, D. Sontag, D. L. Ong and A. Kolobov, “blog: Probabilistic models with unknown objects”, Statistical relational learning p. 373 (2007).

Moore, R. C., “A formal theory of knowledge and action”, Tech. rep., SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER (1984).

Morgenstern, L., “A first order theory of planning, knowledge, and action”, in “Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge”, TARK ’86, pp. 99–114 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1986), URL `http://dl.acm.org/citation.cfm?id=1029786.1029796`.

Nath, A. and P. Domingos, “A language for relational decision theory”, in “Proceedings of the International Workshop on Statistical Relational Learning”, (2009).

Ng, R. and V. Subrahmanian, “Stable semantics for probabilistic deductive databases”, Information and computation **110**, 1, 42–83 (1994).

Nickles, M., “Distribution-aware sampling of answer sets”, in “Scalable Uncertainty Management - 12th International Conference, SUM 2018, Milan, Italy, October 3-5, 2018, Proceedings”, pp. 164–180 (2018), URL `https://doi.org/10.1007/978-3-030-00461-3\_12`.

Nickles, M. and A. Mileo, “Probabilistic inductive logic programming based on answer set programming”, in “15th International Workshop on Non-Monotonic Reasoning (NMR 2014)”, (2014).

Niemelä, I. and P. Simons, “Extending the Smodels system with cardinality and weight constraints”, in “Logic-Based Artificial Intelligence”, edited by J. Minker, pp. 491–521 (Kluwer, 2000).

Niepert, M., J. Noessner and H. Stuckenschmidt, “Log-linear description logics”, in “IJCAI”, pp. 2153–2158 (2011).

Pearl, J., *Causality: models, reasoning and inference*, vol. 29 (Cambridge Univ Press, 2000).

Pelov, N., M. Denecker and M. Bruynooghe, “Well-founded and stable semantics of logic programs with aggregates”, TPLP **7**, 3, 301–353 (2007).

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

Poon, H. and P. Domingos, “Sound and efficient inference with probabilistic and deterministic dependencies”, in “AAAI”, vol. 6, pp. 458–463 (2006).

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

Saad, E. and E. Pontelli, “Hybrid probabilistic logic programming with non-monotoic negation”, in “In Twenty First International Conference on Logic Programming”, (Springer Verlag, 2005).

Sanner, S., *First-order decision-theoretic planning in structured relational environments*, Ph.D. thesis, Citeseer (2008).

Sanner, S., “Relational dynamic influence diagram language (RDDL): Language description”, Unpublished ms. Australian National University p. 32 (2010).

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

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

Shterionov, D. S., A. Kimmig, T. Mantadelis and G. Janssens, "Dnf sampling for problog inference", arXiv preprint arXiv:1009.3798 (2010).

Singla, P. and P. M. Domingos, "Lifted first-order belief propagation.", in "AAAI", vol. 8, pp. 1094–1099 (2008).

Skarlatidis, A., G. Paliouras, G. A. Vouros and A. Artikis, "Probabilistic event calculus based on markov logic networks", in "Rule-Based Modeling and Computing on the Semantic Web", pp. 155–170 (Springer, 2011).

Son, T. C., E. Pontelli and P. H. Tu, "Answer sets for logic programs with arbitrary abstract constraint atoms", in "Proceedings, The Twenty-First National Conference on Artificial Intelligence (AAAI)", (2006).

Sridharan, M., M. Gelfond, S. Zhang and J. Wyatt, "REBA: A refinement-based architecture for knowledge representation and reasoning in robotics", (2015).

Srivastava, S., S. Russell, P. Ruan and X. Cheng, "First-order open-universe pomdps", in "Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference, UAI 2014", edited by N. Zhang and J. Tian, pp. 742–751 (AUAI Press, 2014).

Van den Broeck, G., I. Thon, M. van Otterlo and L. De Raedt, "DTProbLog: A decision-theoretic probabilistic prolog.", vol. 2 (2010).

Van Gelder, A., K. Ross and J. Schlipf, "The well-founded semantics for general logic programs", Journal of ACM **38**, 3, 620–650 (1991).

Vennekens, J., M. Denecker and M. Bruynooghe, "Representing causal information about a probabilistic process", in "Logics In Artificial Intelligence", pp. 452–464 (Springer, 2006).

Vennekens, J., M. Denecker and M. Bruynooghe, "CP-logic: A language of causal probabilistic events and its relation to logic programming", Theory and practice of logic programming **9**, 3, 245–308 (2009).

Vennekens, J., S. Verbaeten and M. Bruynooghe, "Logic programs with annotated disjunctions", in "Logic Programming", edited by B. Demoen and V. Lifschitz, pp. 431–445 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004).

Vlasselaer, J., J. Renkens, G. Van den Broeck and L. De Raedt, "Compiling probabilistic logic programs into sentential decision diagrams", (2014).

Wang, B., Z. Zhang, H. Xu and J. Shen, "Splitting an lpmln program", in "AAAI", (2018).

Wang, C., S. Joshi and R. Khardon, "First order decision diagrams for relational mdps", Journal of Artificial Intelligence Research **31**, 431–472 (2008).

Watkins, C. J. C. H., *Learning from delayed rewards*, Ph.D. thesis (1989).

Yang, F., D. Lyu, B. Liu and S. Gustafson, "PEORL: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making", in "IJCAI", pp. 4860–4866 (2018).

Younes, H. L. and M. L. Littman, "PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects", Techn. Rep. CMU-CS-04-162 (2004).

Zhang, S. and P. Stone, "CORPP: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot", in "Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence", AAAI'15, pp. 1394–1400 (AAAI Press, 2015), URL `http://dl.acm.org/citation.cfm?id=2887007.2887200`.

Zhu, W., *PLOG: Its Algorithms and Applications*, Ph.D. thesis, Texas Tech University (2012).