*TLP* **18** (3–4): 607–622, 2018. © Cambridge University Press 2018
doi:10.1017/S1471068418000303


# *A Probabilistic Extension of Action Language $\mathcal{BC}$+*

JOOHYUNG LEE and YI WANG
*School of Computing, Informatics and Decision Systems Engineering*
*Arizona State University, Tempe, USA*
(*e-mails:* joolee@asu.edu; ywang485@asu.edu)

*submitted 2 May 2018; accepted 13 May 2018*

**Abstract**

We present a probabilistic extension of action language $\mathcal{BC}$+. Just like $\mathcal{BC}$+ is defined as a high-level notation of answer set programs for describing transition systems, the proposed language, which we call $p\mathcal{BC}$+, is defined as a high-level notation of LP$^{\mathrm{MLN}}$ programs—a probabilistic extension of answer set programs. We show 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$^{\mathrm{MLN}}$.

## 1 Introduction

Action languages, such as $\mathcal{A}$ (Gelfond and Lifschitz 1993), $\mathcal{B}$ (Gelfond and Lifschitz 1998), $\mathcal{C}$ (Giunchiglia and Lifschitz 1998), $\mathcal{C}$+ (Giunchiglia *et al*. 2004), and $\mathcal{BC}$ (Lee and Meng 2013), are formalisms for describing actions and their effects. Many of these languages can be viewed as high-level notations of answer set programs structured to represent transition systems. The expressive possibility of action languages, such as indirect effects, triggered actions, and additive fluents, has been one of the main research topics. Most of such extensions are logic-oriented, and less attention has been paid to probabilistic reasoning, with a few exceptions such as (Baral *et al*. 2002; Eiter and Lukasiewicz 2003), let alone automating such probabilistic reasoning and learning parameters of an action description.

Action language $\mathcal{BC}$+ (Babb and Lee 2015), one of the most recent additions to the family of action languages, is no exception. While the language is highly expressive to embed other action languages, such as $\mathcal{C}$+ (Giunchiglia *et al*. 2004) and $\mathcal{BC}$ (Lee *et al*. 2013), it does not have a natural way to express the probabilities of histories (i.e., a sequence of transitions).

In this paper, we present a probabilistic extension of $\mathcal{BC}$+, which we call $p\mathcal{BC}$+. Just like $\mathcal{BC}$+ is defined as a high-level notation of answer set programs for describing transition systems, $p\mathcal{BC}$+ is defined as a high-level notation of LP$^{\mathrm{MLN}}$ programs—a probabilistic extension of answer set programs. Language $p\mathcal{BC}$+ inherits expressive logical modeling capabilities of $\mathcal{BC}$+ but also allows us to assign a probability to a sequence of transitions so that we may distinguish more probable histories.

We show how probabilistic reasoning about transition systems, such as prediction, postdiction, and planning problems, can be modeled in $p\mathcal{BC}$+ and computed using an implementation of LP$^{\mathrm{MLN}}$. Further, we show that it can be used for probabilistic abductive reasoning about dynamic domains, where the likelihood of the abductive explanation is derived from the parameters manually specified or automatically learned from the data.

The paper is organized as follows. Section 2 reviews language LP$^{\mathrm{MLN}}$ and multi-valued probabilistic programs that are defined in terms of LP$^{\mathrm{MLN}}$. Section 3 presents language $p\mathcal{BC}$+, and Section 4 shows how to use $p\mathcal{BC}$+ and system LPMLN2ASP (Lee *et al*. 2017) to perform probabilistic reasoning about transition systems, such as prediction, postdiction, and planning. Section 5 extends $p\mathcal{BC}$+ to handle probabilistic diagnosis.

## 2 Preliminaries

### 2.1 Review: Language LP$^{\mathrm{MLN}}$

We review the definition of LP$^{\mathrm{MLN}}$ from (Lee and Wang 2016; Lee and Wang 2015), limited to the propositional case. An LP$^{\mathrm{MLN}}$ program is a finite set of weighted rules $w : R$ where $R$ is a propositional formula, $w$ is a real number (in which case, the weighted rule is called *soft*) or $\alpha$ for denoting the infinite weight (in which case, the weighted rule is called *hard*).

For any LP$^{\mathrm{MLN}}$ program $\Pi$ and any interpretation $I$, $\overline{\Pi}$ denotes the usual (unweighted) ASP program obtained from $\Pi$ by dropping the weights, and $\Pi_I$ denotes the set of $w : R$ in $\Pi$ such that $I \models R$, and SM[$\Pi$] denotes the set $\{I \mid I \text{ is a stable model of } \overline{\Pi_I}\}$. The *unnormalized weight* of an interpretation $I$ under $\Pi$ is defined as

$$W_\Pi(I) = \begin{cases} exp\left(\sum\limits_{w:R \in \Pi_I} w\right) & \text{if } I \in \mathrm{SM}[\Pi]; \\ 0 & \text{otherwise.} \end{cases}$$

The *normalized weight* (a.k.a. *probability*) of an interpretation $I$ under $\Pi$ is defined as

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

Interpretation $I$ is called a *(probabilistic) stable model* of $\Pi$ if $P_\Pi(I) \neq 0$. The most probable stable models of $\Pi$ are the stable models with the highest probability.

### 2.2 Review: Multi-Valued Probabilistic Programs

Multi-valued probabilistic programs (Lee and Wang 2016) are a simple fragment of LP$^{\mathrm{MLN}}$ that allows us to represent probability more naturally.

We assume that the propositional signature $\sigma$ 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 $\sigma$ 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 *non-probabilistic* 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:

$$p_1 :: \ c{=}v_1 \mid \cdots \mid p_n :: \ c{=}v_n \tag{1}$$

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

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

one for each probabilistic constant $c$, where $\{v_1, \ldots, v_n\} = \mathit{Dom}(c)$, $v_i \neq v_j$, $0 \leq p_1, \ldots, p_n \leq 1$ 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" $c$.

- $\Pi$ is a set of rules of the form *Head* $\leftarrow$ *Body* (identified with formula *Body* $\rightarrow$ *Head* such that *Head* and *Body* do not contain implications, and *Head* contains no probabilistic constants.

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

- For each probabilistic constant declaration (1), $T(\mathbf{\Pi})$ 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) $\alpha :\ \bot \leftarrow c{=}v_i$ if $p_i = 0$.
- For each rule *Head* $\leftarrow$ *Body* in $\Pi$, $T(\mathbf{\Pi})$ contains $\alpha :$ *Head* $\leftarrow$ *Body*.
- For each constant $c$, $T(\mathbf{\Pi})$ contains the uniqueness of value constraints

$$\alpha : \bot \leftarrow c{=}v_1 \wedge c = v_2 \tag{2}$$

for all $v_1, v_2 \in \mathit{Dom}(c)$ such that $v_1 \neq v_2$, and the existence of value constraint

$$\alpha : \bot \leftarrow \neg \bigvee_{v \in Dom(c)} c{=}v\ . \tag{3}$$

In the presence of the constraints (2) and (3), assuming $T(\mathbf{\Pi})$ has at least one (probabilistic) stable model that satisfies all the hard rules, a (probabilistic) stable model $I$ satisfies $c = v$ for exactly one value $v$, so we may identify $I$ with the value assignment that assigns $v$ to $c$.

## 3 Probabilistic $\mathcal{BC}$+

### 3.1 Syntax

We assume a propositional signature $\sigma$ as defined in Section 2.2. 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

$$\textbf{caused } F \textbf{ if } G \tag{4}$$

where $F$ and $G$ are fluent formulas.

A *fluent dynamic law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{5}$$

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.

A *pf constant declaration* is an expression of the form

$$\textbf{caused } c = \{v_1 : p_1, \ldots, v_n : p_n\} \tag{6}$$

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

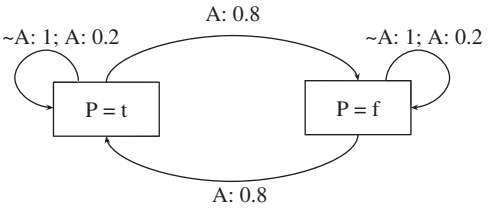


Fig. 1. A transition system with probabilistic transitions

where $c$ is a pf constant with domain $\{v_1, \ldots, v_n\}$, $0 < p_i < 1$ for each $i \in \{1, \ldots, n\}$[2], and $p_1 + \cdots + p_n = 1$. In other words, (6) describes the probability distribution of $c$.

An *initpf constant declaration* is an expression of the form (6) where $c$ is an initpf constant.

An *initial static law* is an expression of the form

$$\textbf{initially } F \textbf{ if } G \tag{7}$$

where $F$ is a fluent constant and $G$ is a formula that contains neither action constants nor pf constants.

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 $\sigma^{fl}$ to denote the set of fluent constants, $\sigma^{act}$ to denote the set of action constants, $\sigma^{pf}$ to denote the set of pf constants, and $\sigma^{initpf}$ to denote the set of initpf constants. For any signature $\sigma'$ and any $i \in \{0, \ldots, m\}$, 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 1*

The following is an action description in $p\mathcal{BC}$+ for the transition system shown in Figure 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** $\{P\}^{\mathrm{ch}}$ **if** $\top$ **after** $P$,
**caused** $\{\sim P\}^{\mathrm{ch}}$ **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** $InitP = x$.

($\{P\}^{\mathrm{ch}}$ is a choice formula standing for $P \vee \neg P$.)

### 3.2 Semantics

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.

For an action description $D$ of a signature $\sigma$, 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

[2] 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.

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

transition system described by $D$. The signature $\sigma_m$ 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 $\sigma_m^x$ to denote the subset of $\sigma_m$

$$\{i : c = v \mid i : c = v \in \sigma_m \text{ and } c \in \sigma^x\}.$$

For $i \in \{0, \ldots, m\}$, we use $i : \sigma^x$ to denote the subset of $\sigma_m^x$

$$\{i : c = v \mid i : c = v \in \sigma_m^x\}.$$

We define $D_m$ to be the multi-valued probabilistic program $\langle PF, \Pi \rangle$, where $\Pi$ is the conjunction of

$$i : F \leftarrow i : G \tag{8}$$

for every static law (4) in $D$ and every $i \in \{0, \ldots, m\}$,

$$i{+}1 : F \leftarrow (i{+}1 : G) \wedge (i : H) \tag{9}$$

for every fluent dynamic law (5) in $D$ and every $i \in \{0, \ldots, m-1\}$,

$$\{0{:}c = v\}^{\mathrm{ch}} \tag{10}$$

for every regular fluent constant $c$ and every $v \in Dom(c)$,

$$\{i : c = \mathbf{t}\}^{\mathrm{ch}}, \quad \{i : c = \mathbf{f}\}^{\mathrm{ch}} \tag{11}$$

for every action constant $c$, and $PF$ consists of

$$p_1 :: \ i : pf = v_1 \mid \cdots \mid p_n :: \ i : pf = v_n \tag{12}$$

($i = 0, \ldots, m-1$) for each pf constant declaration (6) in $D$ that describes the probability distribution of *pf*.

Also, 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 $\Pi^{init}$ consists of the rule

$$\bot \leftarrow \neg(0{:}F) \wedge 0{:}G$$

for each initial static law (7), and $PF^{init}$ consists of

$$p_1 :: \ 0{:}pf = v_1 \ \mid \ \ldots \ \mid \ p_n :: \ 0{:}pf = v_n$$

for each initpf constant declaration (6).

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 2*

For the action description *PSD* in Example 1, $PSD_{init}$ is the following multi-valued probabilistic program ($x \in \{\mathbf{t}, \mathbf{f}\}$):

$$0.6 :: \ 0{:}\mathit{InitP} \ \mid \ 0.4 :: \ 0{:}{\sim}\mathit{InitP}$$
$$\bot \leftarrow \neg(0{:}P{=}x) \wedge 0 : \mathit{InitP}{=}x.$$

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

and $PSD_m$ is the following multi-valued probabilistic program ($i$ is a schematic variable that ranges over $\{1,\dots,m-1\}$):

$$0.8 :: \ i : Pf \ \mid \ 0.2 :: \ i :\sim Pf$$
$$i{+}1 : P \leftarrow i :\sim P \wedge i : A \wedge i : Pf$$
$$i{+}1 :\sim P \leftarrow i : P \wedge i : A \wedge i : Pf$$
$$\{i{+}1 : P\}^{\mathrm{ch}} \leftarrow i : P$$
$$\{i{+}1 :\sim P\}^{\mathrm{ch}} \leftarrow i :\sim P$$

$$\{i : A\}^{\mathrm{ch}}$$
$$\{i :\sim A\}^{\mathrm{ch}}$$
$$\{0 : P\}^{\mathrm{ch}}$$
$$\{0 :\sim P\}^{\mathrm{ch}}$$

For any LP$^{\mathrm{MLN}}$ program $\Pi$ of signature $\sigma$ and a value assignment $I$ to a subset $\sigma'$ of $\sigma$, we say $I$ is a *residual (probabilistic) stable model* of $\Pi$ if there exists a value assignment $J$ to $\sigma \setminus \sigma'$ such that $I \cup J$ is a (probabilistic) stable model of $\Pi$.

For any value assignment $I$ to constants in $\sigma$, 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 $\sigma^{fl}$ 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 $\sigma^{fl}$ and $e$ is an interpretation of $\sigma^{act}$ 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 $\sigma^{pf}$ 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 1*
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 $\sigma^{act}$ such that at most one action is true, and any value assignment $pf$ of $\sigma^{pf}$, 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 $\sigma^{initpf}$, there exists exactly one assignment $fl$ of $\sigma^{fl}$ 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 $\sigma^{act}$ such that at most one action is true, and any value assignment $pf$ of $\sigma^{pf}$, 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 $\sigma'$ of $\sigma$, 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 $\sigma_m^{act}$, 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 $\sigma_m^{act}$, we have $I_{TC\cup A}(p) = A(p)$;

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

- $(I_{TC\cup A})|_{0:\sigma^{fl}}$ 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 $i\in\{1,\ldots,m\}$,

$$(I_{TC\cup A})|_{i:\sigma^{fl}} = \phi((I_{TC\cup A})|_{(i-1):\sigma^{fl}}, (I_{TC\cup A})|_{(i-1):\sigma^{act}}, (I_{TC\cup A})|_{(i-1):\sigma^{pf}}).$$

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 $0:\sigma^{initpf}\cup\sigma_m^{pf}$. We thus call the value assignment of an interpretation $I$ on $0:\sigma^{initpf}\cup\sigma_m^{pf}$ 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 1*
For any value assignment $TC$ of $0:\sigma^{initpf}\cup\sigma_m^{pf}$ and any value assignment $A$ of $\sigma_m^{act}$, 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

$$Pr_{Tr(D,m)}(I_{TC\cup A}) = \frac{\prod_{c=v\in TC} M(c=v)}{(|\sigma^{act}|+1)^m}.$$

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 2*
For any state $s$ and $s'$, and any interpretation $e$ of $\sigma^{act}$, 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 $i,j\in\{0,\ldots,m-1\}$ such that $Pr_{Tr(D,m)}(i:s)>0$ and $Pr_{Tr(D,m)}(j:s)>0$.

For every subset $X_m$ of $\sigma_m\setminus\sigma_m^{pf}$, 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 $\sigma_0^{fl}$ 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 1*
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,

$$Pr_{Tr(D,m)}(X_m \mid 0:e_0,\ldots,m-1:e_{m-1}) = p(X^0)\times\cdots\times p(X^m)\times Pr_{Tr(D,m)}(0:s_0).$$

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

*Example 3*

Consider the simple transition system with probabilistic effects in Example 1. Suppose $a$ is executed twice. What is the probability that $P$ remains true the whole time? Using Corollary 1 this can be computed as follows:

$$\begin{aligned}&Pr(2:P=\mathbf{t},1:P=\mathbf{t},0:P=\mathbf{t}\mid 0:A=\mathbf{t},1:A=\mathbf{t})\\&=p(\langle P=\mathbf{t},A=\mathbf{t},P=\mathbf{t}\rangle)\cdot p(\langle P=\mathbf{t},A=\mathbf{t},P=\mathbf{t}\rangle)\cdot Pr_{Tr(D,m)}(0:P=\mathbf{t})\\&=\ 0.2\times 0.2\times 0.6=0.024.\end{aligned}$$

## 4 $p\mathcal{BC}$+ 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 (Lee *et al*. 2017). Consider the following probabilistic variation of the well-known Yale Shooting Problem: There are two (slightly 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:

**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

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

**caused** $\sim$*Alive*$(t)$ **if** $\top$ **after** *Loaded* $\wedge$ *Fire*$(t)\wedge$ $\sim$*Alert*$(t)$ $\wedge$ *Pf_Killed*$(t)$,
**caused** $\sim$*Alive*$(t)$ **if** $\top$ **after** *Loaded* $\wedge$ *Fire*$(t)$ $\wedge$ *Alert*$(t)$ $\wedge$ *Pf_Killed_Alert*$(t)$,
**caused** $\sim$*Loaded* **if** $\top$ **after** *Fire*$(t)$.

A dead turkey causes the other turkey to be alert:

**default** $\sim$*Alert*$(t)$,
**caused** *Alert*$(t_1)$ **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:

**caused** $\{$*Alive*$(t)\}^{\text{ch}}$ **if** $\top$ **after** *Alive*$(t)$,
**caused** $\{\sim$*Alive*$(t)\}^{\text{ch}}$ **if** $\top$ **after** $\sim$*Alive*$(t)$,
**caused** $\{$*Loaded*$\}^{\text{ch}}$ **if** $\top$ **after** *Loaded*,
**caused** $\{\sim$*Loaded*$\}^{\text{ch}}$ **if** $\top$ **after** $\sim$*Loaded*.

We ensure no concurrent actions are allowed by stating

**caused** $\bot$ **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 $\{\mathbf{t}, \mathbf{f}\}$):

**caused** *Init_Alive*$(t) = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\}$,
**caused** *Init_Loaded* $= \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\}$,
**initially** *Alive*$(t) = b$ **if** *Init_Alive*$(t) = b$,
**initially** *Loaded* $= b$ **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.[3]

**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 dies 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:

[3] The complete LPMLN2ASP program and the queries used in this section are given in Appendix B of the supplementary material corresponding to this paper at the TPLP archives (Lee and Wang 2018).

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

```
:- 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
```

**Planning** In this case, we are interested in a sequence of actions that would result in the highest probability of a certain goal. Formally, we are interested in

$$\underset{Act}{\text{argmax}} \; Pr_{Tr(D,m)}(Goal \mid Initial\_State, Act)$$

where $Goal$ is a condition for a goal state, and $Act$ is a sequence of actions $a \in \sigma^{act}$ specifying actions executed at each timestep.

In the Yale shooting example, such query can be "given that both turkeys are alive and the gun is not loaded at the beginning, generate a plan that gives best chance to kill both the turkeys with 4 actions."

Formalizing the query and executing the command

```
lpmln2asp -i yale-shooting.lpmln
```

finds the most probable stable model, which yields

```
load(t,0)    fire(slimTurkey,t,1)
load(t,2)    fire(fatTurkey,t,3)
```

which suggests to first kill the slim turkey and then the fat turkey.

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

## 5 Diagnosis in Probabilistic Action Domain

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.

### *5.1 Extending $p\mathcal{BC}$+ to Allow Diagnosis*

We define the following new constructs to allow probabilistic diagnosis in action domains. Note that these 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 the Boolean domain, and add

  **default** $\sim ab$.

- We introduce the expression

  **caused_ab** $F$ **if** $G$ **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

  **caused** $F$ **if** $ab \wedge G$ **after** $H$.

Once we have defined abnormalities and how they affect the system, we can use

**caused** $ab$

to enable taking abnormalities into account in reasoning.

### *5.2 Example: Robot*

The following example is modified from (Iwan 2002). Consider a robot located in a building with two 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. There is a $0.1$ chance that it fails when it tries to enter a room, a $0.2$ chance that the robot drops the book when it has the book, and a $0.3$ chance that the robot fails when it tries to pick up the book. The robot, as well as the book, was initially at `r1`. It 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 is a possible reason?

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

We answer this query by modeling the action domain in the probabilistic action language as follows. We first introduce the following constants.

| | |
|---|---|
| Notation: $r$ range over $\{R_1, R_2\}$. | |
| Regular fluent constants: | Domains: |
| *LocRobot*, *LocBook* | $\{R_1, R_2\}$ |
| *HasBook* | Boolean |
| Abnormal fluent constants: | Domains: |
| *EnterFailed*, *DropBook*, *PickupFailed* | Boolean |
| Action constants: | Domains: |
| *Goto*$(r)$, *PickUpBook*, *PutdownBook* | Boolean |
| Pf constants: | Domains: |
| *Pf_EnterFailed*, *Pf_PickupFailed*, *Pf_DropBook* | Boolean |
| Initpf constants: | Domains: |
| *Init_LocRobot*, *Init_LocBook* | $\{R_1, R_2\}$ |
| *Init_HasBook* | Boolean |

The action $Goto(r)$ causes the location of the robot to be at $r$ unless the abnormality *EnterFailed* occurs:

**caused** *LocRobot*$=r$ **after** *Goto*$(r)$ $\wedge \neg$*EnterFailed*.

Similarly, the following causal laws describe the effect of the actions *PickupBook* and *PutdownBook*:

**caused** *HasBook* **if** *LocRobot* $=$ *LocBook* **after** *PickUpBook* $\wedge \neg$*PickUpFailed*
**caused** $\sim$*HasBook* **after** *PutdownBook*.

If the robot has the book, then the book has the same location as the robot:

**caused** *LocBook* $= r$ **if** *LocRobot* $= r \wedge$ *HasBook*.

The abnormality *DropBook* causes the robot to not have the book:

**caused** $\sim$*HasBook* **if** *DropBook*.

The fluents *LocBook*, *LocRobot* and *HasBook* observe the commonsense law of inertia:

**caused** $\{$*LocBook* $= r\}^{\text{ch}}$ **after** *LocBook* $= r$
**caused** $\{$*LocRobot* $= r\}^{\text{ch}}$ **after** *LocRobot* $= r$
**caused** $\{$*HasBook* $= b\}^{\text{ch}}$ **after** *HasBook* $= b$.

The abnormality *EnterFailed* has $0.1$ chance to occur when the action $Goto$ is executed:

**caused** $\{\sim$*EnterFailed*$\}^{\text{ch}}$ **if** $\sim$*EnterFailed*
**caused** *Pf_EnterFailed* $= \{\mathbf{t} : 0.1, \mathbf{f} : 0.9\}$
**caused_ab** *EnterFailed* **if** $\top$ **after** *pf_EnterFailed* $\wedge$ *Goto*$(r)$.

Similarly, the following causal laws describe the condition and probabilities for the abnormalities *PickupFailed* and *DropBook* to occur:

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

**caused** $\{\sim PickupFailed\}^{\text{ch}}$ **if** $\sim PickupFailed$
**caused** $Pf\_PickupFailed = \{\mathbf{t} : 0.3, \mathbf{f} : 0.7\}$
**caused_ab** $PickupFailed$ **if** $\top$ **after** $Pf\_PickupFailed \wedge PickupBook$,

**caused** $\{\sim DropBook\}^{\text{ch}}$ **if** $\sim DropBook$
**caused** $Pf\_DropBook = \{\mathbf{t} : 0.2, \mathbf{f} : 0.8\}$
**caused_ab** $DropBook$ **if** $\top$ **after** $Pf\_DropBook \wedge HasBook$.

We ensure no concurrent actions are allowed by stating

**caused** $\bot$ **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:

**caused** $Init\_LocRobot = \{R_1 : 0.5, R_2 : 0.5\}$
**caused** $Init\_LocBook = \{R_1 : 0.5, R_2 : 0.5\}$
**caused** $Init\_HasBook = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\}$
**initially** $LocRobot = r$ **if** $Init\_LocRobot = r$
**initially** $LocBook = r$ **if** $Init\_LocBook = r$
**initially** $HasBook = b$ **if** $Init\_HasBook = b$.

No abnormalities are possible in 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, together with the actions that the robot has executed.[4]

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

[4] For the complete translation of the action description in the language of LPMLN2ASP, we refer the reader to Appendix C of the supplementary material corresponding to this paper at the TPLP archives (Lee and Wang 2018).

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

```
:- 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`.

## 6 Related Work

There exist various formalisms for reasoning in probabilistic action domains. $PC+$ (Eiter and Lukasiewicz 2003) is a generalization of the action language $\mathcal{C}+$ that allows for expressing probabilistic information. The syntax of $PC+$ is similar to $p\mathcal{BC}+$, as both the languages are extensions of $\mathcal{C}+$. $PC+$ 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 $\text{LP}^{\text{MLN}}$, 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, $PC+$ 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 $PC+$ 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 $PC+$, 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 PDDL 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.

https://doi.org/10.1017/S1471068418000303 Published online by Cambridge University Press

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$^{\mathrm{MLN}}$ systems, its support for probabilistic diagnosis, and the possibility of parameter learning in actions domains.

LP$^{\mathrm{MLN}}$ is closely related to Markov Logic Networks (Richardson and Domingos 2006), a formalism originating from Statistical Relational Learning. However, Markov Logic Networks have not been applied to modeling dynamic domains due to its limited expressivity from its logical part.

## 7 Conclusion

$p\mathcal{BC}$+ is a simple extension of $\mathcal{BC}$+. The main idea is to assign a probability to each path of a transition system to distinguish the likelihood of the paths. The extension is a natural composition of the two ideas: In the semantics of $\mathcal{BC}$+, the paths are encoded as stable models of the logic program standing for the $\mathcal{BC}$+ description. Since LP$^{\mathrm{MLN}}$ is a probabilistic extension of ASP, it comes naturally that by lifting the translation to turn into LP$^{\mathrm{MLN}}$ we could achieve a probabilistic action language.

In the examples above, the action descriptions, including the probabilities, are all hand-written. In practice, the exact values of some probabilities are hard to find. In particular, it is not likely to have a theoretical probability for an abnormality to occur. It is more practical to statistically derive the probability from a collection of action and observation histories. For example, in the robot example in Section 5.2, we can provide a list of action and observation histories, where different abnormalities occurred, as the training data. With this training data, we may learn the weights of the LP$^{\mathrm{MLN}}$ rules that control the probabilities of abnormalities.

Another future work is to build a compiler that automates the process of the translation of $p\mathcal{BC}$+ description into the input language of LPMLN2ASP by extending a system like CPLUS2ASP (Babb and Lee 2013).

## Acknowledgements

We are grateful to Zhun Yang and the anonymous referees for their useful comments. This work was partially supported by the National Science Foundation under Grant IIS-1526301.

## Supplementary materials

For supplementary material for this article, please visit https://doi.org/10.1017/S1471068418000303

## References

BABB, J. AND LEE, J. 2013. Cplus2ASP: Computing action language C+ in answer set programming. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. 122–134.

BABB, J. AND LEE, J. 2015. Action language BC+. *Journal of Logic and Computation*, exv062.

BALDUCCINI, M. AND GELFOND, M. 2003. Diagnostic reasoning with A-Prolog. *Theory and Practice of Logic Programming 3*, 425–461.

BARAL, C., GELFOND, M., AND RUSHTON, N. 2004. Probabilistic reasoning with answer sets. In *Logic Programming and Nonmonotonic Reasoning*. Springer Berlin Heidelberg, Berlin, Heidelberg, 21–33.

BARAL, C., MCILRAITH, S., AND SON, T. 2000. Formulating diagnostic problem solving using an action language with narratives and sensing.

BARAL, C., TRAN, N., AND TUAN, L.-C. 2002. Reasoning about actions in a probabilistic setting. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 507–512.

D'ASARO, F. A., BIKAKIS, A., DICKENS, L., AND MILLER, R. 2017. Foundations for a probabilistic event calculus. *CoRR abs/1703.06815*.

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

GELFOND, M. AND LIFSCHITZ, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming 17*, 301–322.

GELFOND, M. AND LIFSCHITZ, V. 1998. Action languages[5]. *Electronic Transactions on Artificial Intelligence 3*, 195–210.

GIUNCHIGLIA, E., LEE, J., LIFSCHITZ, V., MCCAIN, N., AND TURNER, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence 153(1–2)*, 49–104.

GIUNCHIGLIA, E. AND LIFSCHITZ, V. 1998. An action language based on causal explanation: Preliminary report. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 623–630.

IWAN, G. 2002. History-based diagnosis templates in the framework of the situation calculus. *AI Communications 15,* 1, 31–45.

LEE, J., LIFSCHITZ, V., AND YANG, F. 2013. Action language BC: Preliminary report. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

LEE, J. AND MENG, Y. 2013. Answer set programming modulo theories and reasoning about continuous changes. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

LEE, J., TALSANIA, S., AND WANG, Y. 2017. Computing LPMLN using ASP and MLN solvers. *Theory and Practice of Logic Programming*.

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

LEE, J. AND WANG, Y. 2016. Weighted rules under the stable model semantics. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 145–154.

LEE, J. AND WANG, Y. 2018. Online appendix for the paper "A probabilistic extension of action language BC+".

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

SKARLATIDIS, A., PALIOURAS, G., VOUROS, G. A., AND ARTIKIS, A. 2011. Probabilistic event calculus based on markov logic networks. In *Rule-Based Modeling and Computing on the Semantic Web*. Springer, 155–170.

YOUNES, H. L. AND LITTMAN, M. L. 2004. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects.

ZHU, W. 2012. Plog: Its algorithms and applications. Ph.D. thesis, Texas Tech University.

[5] http://www.ep.liu.se/ea/cis/1998/016/