# Probabilistic Action Language $p\mathcal{BC}+$

**Yi Wang**
Arizona State University
School of Computing, Informatics, and Decision Systems Engineering
Fulton Schools of Engineering, Arizona State University
P.O. Box 878809, Tempe, AZ 85287-8809, United States
ywang485@asu.edu

## Abstract

We present an ongoing research on 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$^{\text{MLN}}$ programs – a probabilistic extension of answer set programs.

As preliminary results accomplished, we illustrate 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$^{\text{MLN}}$.

For future work, we plan to develop a compiler that automatically translates $p\mathcal{BC}+$ description into LP$^{\text{MLN}}$ programs, as well as parameter learning in probabilistic action domains through LP$^{\text{MLN}}$ weight learning. We will work on defining useful extensions of $p\mathcal{BC}+$ to facilitate hypothetical/counterfactual reasoning. We will also find real-world applications, possibly in robotic domains, to empirically study the performance of this approach to probabilistic reasoning in action domains.

**2012 ACM Subject Classification** Computing methodologies → Knowledge representation and reasoning

**Keywords and phrases** action language, probabilistic reasoning, LP$^{\text{MLN}}$

**Digital Object Identifier** 10.4230/OASIcs.ICLP.2018.15

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

## 1 Introduction and Problem Description

Action languages, such as $\mathcal{A}$ [9], $\mathcal{B}$ [10], $\mathcal{C}$ [12], $\mathcal{C}+$ [11], and $\mathcal{BC}$ [15], 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 the extensions accounting for that are logic-oriented, and less attention has been paid to probabilistic reasoning, with a few exceptions such as [6, 8], let alone automating such probabilistic reasoning and learning parameters of an action description.

Action language $\mathcal{BC}+$ [2], 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 $\mathcal{C}+$ [11] and $\mathcal{BC}$ [14], it does not have a natural way to express the likelihood of histories (i.e., a sequence of transitions).


Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018).
Editors: Alessandro Dal Palu', Paul Tarau, Neda Saeedloei, and Paul Fodor; Article No. 15; pp. 15:1–15:12
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Example 1.** Consider an extension of the robot example from [13]: A robot and a book that can be picked up are located in a building with 2 rooms `r1` and `r2`. The robot can move to rooms, pick up the book and put down the book. There is 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 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 was the problem?

To answer the above query, an action language needs the capabilities of not only probabilistic reasoning, but also abductive reasoning in a probabilistic setting. In my research, we are working on a probabilistic extension of $\mathcal{BC}+$, which we call $p\mathcal{BC}+$, with the expressivity to answer queries such as the one in Example 1. 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$^{\text{MLN}}$ 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.

In this paper, as preliminary results accomplished, we will 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$^{\text{MLN}}$[16]. Further, we will 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.

For future work, we plan to develop a compiler that automatically translates $p\mathcal{BC}+$ description into LP$^{\text{MLN}}$ programs, as well as parameter learning in probabilistic action domains through LP$^{\text{MLN}}$ weight learning. We will work on defining useful extensions of $p\mathcal{BC}+$ to facilitate hypothetical/counterfactual reasoning. We will also find real-world applications, possibly in robotic domains, to empirically study the performance of this approach to probabilistic reasoning in action domains.

This paper will give a summary of my research on $p\mathcal{BC}+$, including the background and some review of existing literature (Section 2), goal of the research (Section 3), the current status of the research (Section 4), preliminary results accomplished (Section 5) as well as issues and expected achievements (Section 6).

## 2 Background and Overview of Existing Literature

### 2.1 Probabilistic Reasoning and Diagnosis in the Context of Action Languages

There are various formalisms for reasoning in probabilistic action domains. $\mathcal{PC}+$ [8] is a generalization of the action language $\mathcal{C}+$ that allows for expressing probabilistic information. $\mathcal{PC}+$ expresses probabilistic transition of states through so-called *context variables*, which are exogenous variables associated with predefined probability distributions. $\mathcal{PC}+$ 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 $\mathcal{PC}+$ as far as we know.

[20] 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 [4] and since there exists a system for computing P-log, reasoning in $\mathcal{NB}$ action descriptions can be automated. Like $\mathcal{PC}+$, 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. In order to translate $\mathcal{NB}$ into executable low-level logic programming languages, some semantical assumptions have to be made in $\mathcal{NB}$, such as all actions have to be always executable and nondeterminism can only be caused by random variables.

Probabilistic action domains, especially in terms of probabilistic effects of actions, can be formalized as Markov Decision Process (MDP). The language proposed in [6] aims at facilitating elaboration tolerant representations of MDPs. The syntax is similar to $\mathcal{NB}$ and $\mathcal{PC}+$. 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 random variables in $\mathcal{NB}$. 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 [19] 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 [7] and [18].

In the above formalisms, the problem of probabilistic diagnosis is only discussed in [20]. [3] and [5] studied the problem of diagnosis. However, they are focused on diagnosis in deterministic and static domains. [13] has proposed a method for diagnosis in action domains with situation calculus. Again, the diagnosis considered there does not involve any probabilistic measure.

## 2.2 Review: Language LP$^{\text{MLN}}$

We review the definition of LP$^{\text{MLN}}$ from [17]. An LP$^{\text{MLN}}$ program is a finite set of weighted rules $w : R$ where $R$ is a rule and $w$ 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*). An LP$^{\text{MLN}}$ program is called *ground* if its rules contain no variables. We assume a finite Herbrand Universe so that the ground program is finite. Each ground instance of a non-ground rule receives the same weight as the original non-ground formula.

For any ground LP$^{\text{MLN}}$ program $\Pi$ and any interpretation $I$, $\overline{\Pi}$ denotes the usual (unweighted) ASP program obtained from $\Pi$ by dropping the weights, $\Pi_I$ 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

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

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

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

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.

### 2.3 Review: Multi-Valued Probabilistic Programs

Multi-valued probabilistic programs [17] are a simple fragment of LP$^{\rm MLN}$ 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:

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

  one for each probabilistic constant $c$, where $\{v_1,\dots,v_n\} = Dom(c)$, $v_i \neq v_j$, $0 \leq p_1,\dots,p_n \leq 1$ 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 such that the head contains no probabilistic constants.

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

- For each probabilistic constant declaration (1), $T(\mathbf{\Pi})$ contains, for each $i = 1,\dots,n$, (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

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

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

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

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 Goal of the Research

The following are our research objectives.

- **Designing Probabilistic Action Language on the Foundation of** LP$^{\rm MLN}$**.** We design the syntax and semantics of the language $p\mathcal{BC}+$ to allow for commonsense reasoning, probabilistic inference and statistical learning. Furthermore, we study the theoretical properties of the action language to establish its relation with probabilistic transition systems.

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

- **Defining the Extension of the Action Language to Explain the Reason of Failure in Dynamic Domains.** We extend the probabilistic action language to account for diagnostic reasoning when the observation conflicts with the way the system is supposed to behave. This will be in contrast with diagnostic reasoning in other action languages, which is logical and does not distinguish which diagnosis is more probable.
- **Extending the Action Language For Hypothetical/Counterfactual Reasoning.** We extend the probabilistic action language to answer queries involving hypothetical/-counterfactual reasoning, where the diagnosis or observation is given, we are interested in how the outcome would have been affected if some action happened instead.
- **Implementing a Compiler that Automatically Translates $p\mathcal{BC}+$ Descriptions to LP$^{\text{MLN}}$ Programs.** Since $p\mathcal{BC}+$ can be executable through translation to LP$^{\text{MLN}}$, it is desirable to have a compiler that automates this translation. We plan to develop such a compiler.
- **Empirically Studying the Performance of $p\mathcal{BC}+$ with Real-World Applications.** After we have the implementation for inference and learning on $p\mathcal{BC}+$ action descriptions, we will apply $p\mathcal{BC}+$ on reasoning and learning tasks in real-world applications, possibly robotic domains.

## 4 Current Status of the Research

This research is at its starting phase. In our recent paper accepted by ICLP 2018, we have defined the syntax and semantics of $p\mathcal{BC}+$, and experimented with several examples through manual translation to LP$^{\text{MLN}}$. We have also defined the extension that allows diagnostic reasoning in probabilistic action domains.

Currently we are investigating on parameter learning of $p\mathcal{BC}+$ through LP$^{\text{MLN}}$ weight learning. We are developing a prototype system for LP$^{\text{MLN}}$ weight learning, and several examples of parameter learning of $p\mathcal{BC}+$ descriptions are part of the benchmarks we use for the prototype system.

## 5 Preliminary Results Accomplished

In this section, we will present the syntax and semantics of $p\mathcal{BC}+$, and illustrate how various reasoning tasks involving probabilistic inference can be automated in this language, through translation to LP$^{\text{MLN}}$.

### 5.1 Syntax of $p\mathcal{BC}+$

We assume a propositional signature $\sigma$ as defined in Section 2.3. 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.

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

$$\textbf{caused } pf = \{v_1 : p_1, \dots, v_n : p_n\} \tag{6}$$

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

An *initpf constant declaration* is an expression of the form (6) where $pf$ 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 formula and $G$ is a formula that contains neither action constant nor pf constant.

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 in $D$. For any signature $\sigma'$ 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 2.** 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 $\{\mathbf{t}, \mathbf{f}\}$.)

$$\begin{array}{ll}
\textbf{caused } P \textbf{ if } \top \textbf{ after } \sim\!P \wedge A \wedge Pf, & \textbf{caused } Pf = \{\mathbf{t} : 0.8, \mathbf{f} : 0.2\}, \\
\textbf{caused } \sim\!P \textbf{ if } \top \textbf{ after } P \wedge A \wedge Pf, & \textbf{caused } Init_P = \{\mathbf{t} : 0.6, \mathbf{f} : 0.4\}, \\
\textbf{caused } \{P\}^{\rm ch} \textbf{ if } \top \textbf{ after } P, & \textbf{initially } P = x \textbf{ if } Init_P = x. \\
\textbf{caused } \{\sim\!P\}^{\rm ch} \textbf{ if } \top \textbf{ after } \sim\!P, &
\end{array}$$

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

## 5.2 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.

---

[2] We require $0 < p_i < 1$ for each $i \in \{1, \dots, n\}$ 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 $pf$ or there is not really a need to introduce $pf$ as a pf constant. So this assumption does not really sacrifice expressivity.

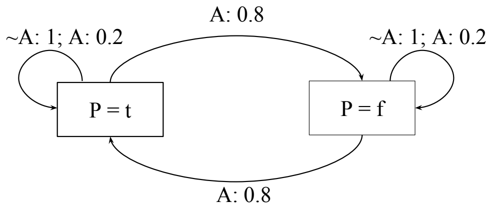


**Figure 1** A transition system with probabilistic transitions.

For an action description $D$ of a signature $\sigma$, we define a sequence of multi-valued probabilistic program $D_0, D_1, \ldots, D_m$ so that the stable models of $D_m$ can be identified with the paths in the 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 $v \in Dom(c)$.

We use $\sigma^x_m$, where $x \in \{act, fl, pf\}$, to denote the subset of $\sigma_m$

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

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 $i \in \{0, \ldots, m-1\}$;

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

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

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

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 $pf$.

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

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

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

$$p_1 :: \ 0\!:\!c = v_1 \mid \cdots \mid p_n :: \ 0\!:\!c = v_n$$

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 3.** For the action description *PSD* in Example 2, $PSD_{init}$ is the following multi-valued probabilistic program ($x \in \{\mathbf{t}, \mathbf{f}\}$):

$$\begin{array}{l} 0.6 :: \ 0\!:\!\mathit{Init\_P} \ \mid \ 0.4 :: \ 0\!:\!\sim\!\mathit{Init\_P} \\ \bot \leftarrow \neg(0\!:\!P\!=\!x) \wedge 0 : \mathit{Init\_P}\!=\!x. \end{array}$$

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

$$\begin{array}{ll} 0.8 :: \ i : Pf \ \mid \ 0.2 :: \ i :\sim\! Pf & \{i\!+\!1 : P\}^{\rm ch} \leftarrow i : P \\ i\!+\!1 : P \leftarrow i :\sim\! P \wedge i : A \wedge i : Pf & \{i\!+\!1 :\sim\! P\}^{\rm ch} \leftarrow i :\sim\! P \\ i\!+\!1 :\sim\! P \leftarrow i : P \wedge i : A \wedge i : Pf & \{i : A\}^{\rm ch} \qquad \{i :\sim\! A\}^{\rm ch} \\ & \{0\!:\!P\}^{\rm ch} \qquad \{0\!:\sim\! P\}^{\rm ch} \end{array}$$

## 5.3 $p\mathcal{BC}+$ Action Descriptions and Probabilistic Reasoning

In this section, we illustrate how the probabilistic extension of the reasoning tasks discussed in [11], i.e., prediction, postdiction and planning, can be represented in $p\mathcal{BC}+$ and automatically computed using LPMLN2ASP [16]. 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:

---

Notation: $t$ range over $\{\mathit{SlimTurkey}, \mathit{FatTurkey}\}$.

| | |
|---|---|
| Regular fluent constants: | Domains: |
| $\mathit{Alive}(t)$, $\mathit{Loaded}$ | Boolean |
| Statically determined fluent constants: | Domains: |
| $\mathit{Alert}(t)$ | Boolean |
| Action constants: | Domains: |
| $\mathit{Load}$ , $\mathit{Fire}(t)$ | Boolean |
| Pf constants: | Domains: |
| $\mathit{Pf\_Killed}(t)$, $\mathit{Pf\_Killed\_Allert}(t)$ | Boolean |
| InitPf constants: | |
| $\mathit{Init\_Alive}(t)$, $\mathit{Init\_Loaded}$ | Boolean |

**caused** $\mathit{Loaded}$ **if** $\top$ **after** $\mathit{Load}$
**caused** $\mathit{Pf\_Killed}(\mathit{SlimTurkey}) = \{\mathbf{t} : 0.6, \mathbf{f} : 0.4\}$
**caused** $\mathit{Pf\_Killed\_Alert}(\mathit{SlimTurkey}) = \{\mathbf{t} : 0.3, \mathbf{f} : 0.7\}$
**caused** $\mathit{Pf\_Killed}(\mathit{FatTurkey}) = \{\mathbf{t} : 0.9, \mathbf{f} : 0.1\}$

**caused** $Pf\_Killed\_Alert(FatTurkey) = \{\mathbf{t} : 0.7, \mathbf{f} : 0.3\}$
**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** $Fire(t)$
**default** $\sim\!Alert(t)$
**caused** $Alert(t_1)$ **if** $\sim\!Alive(t_2) \wedge Alive(t_1) \wedge t_1 \neq t_2$
**caused** $\{Alive(t)\}^{\text{ch}}$ **if** $\top$ **after** $Alive(t)$,
**caused** $\{Loaded\}^{\text{ch}}$ **if** $\top$ **after** $Loaded$
**caused** $\{\sim\!Alive(t)\}^{\text{ch}}$ **if** $\top$ **after** $\sim\!Alive(t)$
**caused** $\{\sim\!Loaded\}^{\text{ch}}$ **if** $\top$ **after** $\sim\!Loaded$
**caused** $\bot$ **after** $a_1 \wedge a_2$
**caused** $Init\_Alive(t) = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\}$ **initially** $Alive(t) = b$ **if** $Init\_Alive(t) = b$
**caused** $Init\_Loaded = \{\mathbf{t} : 0.5, \mathbf{f} : 0.5\}$ **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.

**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 inference $\underset{Result}{\text{argmax}} Pr_{Tr(D,m)}(Result \mid Act, Obs)$, 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 $v \in Dom(c)$.

For example, 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 inference $\underset{Initial\_State}{\text{argmax}} Pr_{Tr(D,m)}(Initial\_State \mid Act, Obs)$, where $Initial\_State$ is a proposition about the initial state; $Act$ and $Obs$ are defined as above.

For example, 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', 1) 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.

For example, in the Yale shooting example, such query can be "given that both the 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.

## 5.4 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 sugar that does 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

  **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.

We can answer the query in Example 1 by modeling the action domain with this extension. Due to lack of space, we skip the details.

# 6 Open Issues and Expected Achievements

The main open issue is that we do not have a compiler that automates the translation from $p\mathcal{BC}+$ to LP$^{\text{MLN}}$. As illustrated in Section 5.3, the action language $p\mathcal{BC}+$ can be executable through translation to LP$^{\text{MLN}}$. It is desirable to have a compiler that automates this translation, so that the user can directly write $p\mathcal{BC}+$ descriptions and does not need to worry about the translation detail. We plan to develop a compiler that translates action descriptions in $p\mathcal{BC}+$ into LP$^{\text{MLN}}$ programs automatically.

The interface and usage of the compiler will be similar to the system CPLUS2ASP [1], which translates the action language $\mathcal{C}+$ to ASP.

Other future works include extending $p\mathcal{BC}+$ for hypothetical/counterfactual reasoning, exploring parameter learning in the setting of probabilistic action language, and empirically studying the performance of $p\mathcal{BC}+$ with weal-world applications.

## References


1. Joseph Babb and Joohyung Lee. Cplus 2ASP: Computing Action Language $\mathcal{C}+$ in Answer Set Programming. In *LPNMR*, 2013.
2. Joseph Babb and Joohyung Lee. Action language $\mathcal{BC}+$. *Journal of Logic and Computation*, page exv062, 2015. `doi:10.1093/logcom/exv062`.
3. Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with A-Prolog. *Theory and Practice of Logic Programming*, 3:425–461, 2003.
4. Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic Reasoning With Answer Sets. In *Logic Programming and Nonmonotonic Reasoning*, pages 21–33, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
5. Chitta Baral, Sheila Mcilraith, and Tran Son. Formulating Diagnostic Problem Solving Using an Action Language With Narratives and Sensing. In *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000.*, pages 311–322, April 2000.
6. Chitta Baral, Nam Tran, and Le-Chi Tuan. Reasoning about actions in a probabilistic setting. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 507–512, 2002.
7. Fabio Aurelio D'Asaro, Antonis Bikakis, Luke Dickens, and Rob Miller. Foundations for a Probabilistic Event Calculus. *CoRR*, abs/1703.06815, 2017. `arXiv:1703.06815`.
8. Thomas Eiter and Thomas Lukasiewicz. Probabilistic Reasoning about Actions in Nonmonotonic Causal Theories. In *Proceedings Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, pages 192–199. Morgan Kaufmann Publishers, 2003.
9. Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.
10. Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 3:195–210, 1998. URL: `http://www.ep.liu.se/ea/cis/1998/016/`.
11. Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.
12. Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 623–630. AAAI Press, 1998.
13. Gero Iwan. History-based diagnosis templates in the framework of the situation calculus. *AI Communications*, 15(1):31–45, 2002.

14 Joohyung Lee, Vladimir Lifschitz, and Fangkai Yang. Action Language $\mathcal{BC}$: Preliminary Report. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

15 Joohyung Lee and Yunsong Meng. Answer Set Programming Modulo Theories and Reasoning about Continuous Changes. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

16 Joohyung Lee, Samidh Talsania, and Yi Wang. Computing LPMLN using ASP and MLN solvers. *Theory and Practice of Logic Programming*, 2017. `doi:10.1017/S1471068417000400`.

17 Joohyung Lee and Yi Wang. Weighted Rules under the Stable Model Semantics. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 145–154, 2016.

18 Anastasios Skarlatidis, Georgios Paliouras, George A Vouros, and Alexander Artikis. Probabilistic event calculus based on markov logic networks. In *Rule-Based Modeling and Computing on the Semantic Web*, pages 155–170. Springer, 2011.

19 Håkan LS Younes and Michael L Littman. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects, 2004.

20 Weijun Zhu. *PLOG: Its Algorithms and Applications*. PhD thesis, Texas Tech University, 2012.