Elaboration Tolerant Representation of Markov Decision Process via Decision-Theoretic Extension of Probabilistic Action Language $p\mathcal{BC}+$

Yi Wang(✉) and Joohyung Lee

School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University, Tempe, USA
{ywang485,joolee}@asu.edu

Abstract. We extend probabilistic action language $p\mathcal{BC}+$ with the notion of utility in decision theory. The semantics of the extended $p\mathcal{BC}+$ can be defined as a shorthand notation for a decision-theoretic extension of the probabilistic answer set programming language LP$^{\text{MLN}}$. Alternatively, the semantics of $p\mathcal{BC}+$ can also be defined in terms of Markov Decision Process (MDP), which in turn allows for representing MDP in a succinct and elaboration tolerant way as well as leveraging an MDP solver to compute a $p\mathcal{BC}+$ action description. The idea led to the design of the system PBCPLUS2MDP, which can find an optimal policy of a $p\mathcal{BC}+$ action description using an MDP solver.

Keywords: Answer set programming · Action language · Markov Decision Process

1 Introduction

Many problems in Artificial Intelligence are about what actions to choose to maximize the agent's utility. Since actions may also have stochastic effects, the main computational task is, 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.

While a few decades of research on action languages has produced several expressive languages, such as $\mathcal{A}$ [5], $\mathcal{B}$ [6], $\mathcal{C}+$ [7], $\mathcal{BC}$ [8], and $\mathcal{BC}+$ [1], that are able to describe actions and their effects in a succinct and elaboration tolerant way, these languages are not equipped with constructs to represent stochastic actions and the utility of a decision. In this paper, we present an action language that overcomes the limitation. Our method is to equip probabilistic action language $p\mathcal{BC}+$ [11] with the notion of utility and define policy optimization problems in that language.

Following the way that $p\mathcal{BC}+$ is defined as a shorthand notation of probabilistic answer set programming language LP$^{\text{MLN}}$ for describing a probabilistic

M. Balduccini et al. (Eds.): LPNMR 2019, LNAI 11481, pp. 224–238, 2019.
https://doi.org/10.1007/978-3-030-20528-7_17

transition system, we first extend LP$^{\text{MLN}}$ by associating a utility measure to each soft stable model in addition to its already defined probability. We call this extension DT-LP$^{\text{MLN}}$. Next, we define a decision-theoretic extension of $p\mathcal{BC}+$ as a shorthand notation for DT-LP$^{\text{MLN}}$. 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 $p\mathcal{BC}+$. 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 leverages MDP solving to find an optimal policy for the $p\mathcal{BC}+$ action description.

The extended $p\mathcal{BC}+$ can thus be viewed as a high-level representation of MDP that allows for compact and elaboration tolerant encodings of sequential decision problems. Compared to other MDP-based planning description languages, such as PPDDL [18] and RDDL [13], it inherits the nonmonotonicity of the stable model semantics to be able to compactly represent recursive definitions and indirect effects of actions, which can save the state space significantly. Section 5 contains such an example.

This paper is organized as follows. After Sect. 2 reviews preliminaries, Sect. 3 extends LP$^{\text{MLN}}$ with the notion of utility, through which we define the extension of $p\mathcal{BC}+$ with utility in Sect. 4. Section 5 defines $p\mathcal{BC}+$ as a high-level representation language for MDP and presents the prototype system PBCPLUS2MDP. We discuss the related work in Sect. 6.

2 Preliminaries

Due to the space limit, the reviews are brief. We refer the reader to the original papers [10,11], or the technical report of this paper [15] for the reviews of preliminaries. The technical report also contains all proofs and experiments with the system PBCPLUS2MDP.

2.1 Review: Action Language $p\mathcal{BC}+$

Like $\mathcal{BC}$ and $\mathcal{BC}+$, language $p\mathcal{BC}+$ assumes that a 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 $c = v$ for every constant c and every element v in $Dom(c)$. If the domain of c is $\{\text{FALSE}, \text{TRUE}\}$, then we say that c is *Boolean*, and abbreviate $c = \text{TRUE}$ as c and $c = \text{FALSE}$ as $\sim c$.

There are four types of constants in $p\mathcal{BC}+$: *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 restricted to Boolean. An *action description*

Causal Laws	**Syntax**	**Translation into** LP^{MLN}
static law	**caused** F **if** G where F and G are fluent formulas	$i:F \leftarrow i:G$ $(i \in \{0,\dots,m\})$
fluent dynamic law	**caused** F **if** G **after** H where F and G are fluent formulas, H is a formula, F does not contain statically determined constants and H does not contain initpf constants	$i{+}1:F \leftarrow (i{+}1:G) \wedge (i:H)$ $(i \in \{0,\dots,m-1\})$
pf constant declaration	**caused** $c = \{v_1 : p_1, \dots, v_n : p_n\}$ where c is a pf constant with domain $\{v_1,\dots,v_n\}$, $0 < p_i < 1$ for each $i \in \{1,\dots,n\}$ and $\sum_{i\in\{1,\dots,n\}} p_i = 1$	For each $j \in \{1,\dots,n\}$: $ln(p_i) : (i{:}c) = v_j$ $(i \in \{0,\dots,m-1\})$
initpf constant declaration	**caused** $c = \{v_1 : p_1, \dots, v_n : p_n\}$ where c is a initpf constant with domain $\{v_1,\dots,v_n\}$, $0 < p_i < 1$ for each $i \in \{1,\dots,n\}$	For each $j \in \{1,\dots,n\}$: $ln(p_i) : (0{:}c) = v_j$
initial static law	**initially** F **if** G where F is a fluent constant and G is a formula that contains neither action constants nor pf constants	$\bot \leftarrow \neg(0{:}F) \wedge 0{:}G$

Fig. 1. Causal laws in $p\mathcal{BC}+$ and their translations into LP^{MLN}

is a finite set of *causal laws*, which describes how fluents depend on each other statically and how their values change from one time step to another. Figure 1 lists causal laws in $p\mathcal{BC}+$ and their translations into LP^{MLN}. A *fluent formula* is a formula such that all constants occurring in it are fluent constants.

We use σ^{fl} (σ^{act}, σ^{pf}, and σ^{initpf}, respectively) to denote the set of all atoms $c = v$ where c is a fluent constant (action constant, pf constant, initpf constant, respectively) of σ and v is in $Dom(c)$. For any subset σ' of σ and any $i \in \{0,\dots,m\}$, we use $i{:}\sigma'$ to denote the set $\{i{:}A \mid A \in \sigma'\}$. For any formula F of signature σ, by $i{:}F$ we denote the result of inserting $i{:}$ in front of every occurrence of every constant in F.

The semantics of a $p\mathcal{BC}+$ action description D is defined by a translation into an LP^{MLN} program $Tr(D,m) = D_{init} \cup D_m$. Below we describe the essential part of the translation that turns a $p\mathcal{BC}+$ description into an LP^{MLN} program.

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, $i \in \{0,\dots,m\}$ and $v \in Dom(c)$,
- for each action constant or pf constant c of D, $i \in \{0,\dots,m-1\}$ and $v \in Dom(c)$.

D_m contains LP^{MLN} rules obtained from static laws, fluent dynamic laws, and pf constant declarations as described in the third column of Fig. 1, as well as $\{0{:}c = v\}^{\text{ch}}$ for every regular fluent constant c and every $v \in Dom(c)$, and $\{i{:}c = \text{TRUE}\}^{\text{ch}}$, $\{i{:}c = \text{FALSE}\}^{\text{ch}}$ $(i \in \{0,\dots,m-1)$ for every action constant c to state that the fluents at time 0 and the actions at each time are exogenous.[1] D_{init} contains LP^{MLN} rules obtained from initial static laws and initpf constant declarations as described in the third column of Fig. 1. Both D_m and D_{init} also contain constraints asserting that each constant is mapped to exactly one value

[1] $\{A\}^{\text{ch}}$ denotes the choice rule $A \leftarrow \mathit{not}\ \mathit{not}\ A$.

in its domain. In the presence of these constraints, we identify an interpretation of σ_m with the value assignment function that maps each constant to its value.

For any LP$^{\text{MLN}}$ program Π of signature σ_1 and an interpretation I of a subset σ_2 of σ_1, we say I is a *residual (probabilistic) stable model* of Π if there exists an interpretation J of $\sigma_1 \setminus \sigma_2$ such that $I \cup J$ is a (probabilistic) stable model of Π.

For any interpretation I of σ, by $i : I$ we denote the interpretation of $i : \sigma$ such that $i : I \models (i : c) = v$ iff $I \models c = v$. For $x \in \{act, fl, pf\}$, we use σ_m^x to denote the subset of σ_m, which is $\{i : c = v \in \sigma_m \mid c = v \in \sigma^x\}$.

A *state* of D is 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 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.

The following simplifying assumptions are made on action descriptions in $p\mathcal{BC}+$.

1. **No Concurrency**: For all transitions $\langle s, e, s' \rangle$, we have $e \models a = \text{TRUE}$ for at most one action constant a;
2. **Nondeterministic Transitions are Determined by pf constants**: For any state s, any value assignment e of σ^{act}, 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 Determined by Initpf constants**: For any value assignment pf_{init} of σ^{initpf}, there exists exactly one value assignment fl of σ^{fl} such that $0 : pf_{init} \cup 0 : fl$ is a stable model of $D_{init} \cup D_0$.

With the above three assumptions, the probability of a history, i.e., a sequence of states and actions, can be computed as the product of the probabilities of all the transitions that the history is composed of, multiplied by the probability of the initial state (Corollary 1 in [11]).

2.2 Review: Markov Decision Process

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) $T : S \times A \times S \to [0, 1]$ defines transition probabilities; (iv) $R : S \times A \times S \to \mathbb{R}$ is the reward function.

Given a history $\langle s_0, a_0, s_1, \ldots, s_{m-1}, a_{m-1}, s_m \rangle$ such that each $s_i \in S$ ($i \in \{0, \ldots, m\}$) and each $a_i \in A$ ($i \in \{0, \ldots, m-1\}$), the *total reward* R_M of the history under MDP M is defined as

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

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

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

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

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

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

Various algorithms for MDP policy optimization have been developed, such as value iteration [3] for exact solutions, and Q-learning [16] for approximate solutions.

3 DT-LP$^{\text{MLN}}$

We extend the syntax and the semantics of LP$^{\text{MLN}}$ to DT-LP$^{\text{MLN}}$ by introducing atoms of the form

$$\texttt{utility}(u, \mathbf{t}) \tag{1}$$

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

$$\alpha : \texttt{utility}(u, \mathbf{t}) \leftarrow \mathit{Body} \tag{2}$$

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

$$U_\Pi(I) = \sum_{\texttt{utility}(u,\mathbf{t})\in I} u.$$

The *expected utility* of a proposition A is defined as

$$E[U_\Pi(A)] = \sum_{I \models A} U_\Pi(I) \times P_\Pi(I \mid A). \tag{3}$$

A DT-LP$^{\text{MLN}}$ program is a pair (Π, Dec) 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 with 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

$$\underset{dec \,:\, dec \text{ is a truth assignment on } Dec}{\operatorname{argmax}} E[U_{\Pi}(dec \wedge e)]. \tag{4}$$

Example 1. 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:

$$\alpha : buy(v) \leftarrow marketTo(v).$$
$$\alpha : buy(v_2) \leftarrow buy(v_1), influence(v_1, v_2).$$
$$\alpha : utility(r, v) \leftarrow buy(v).$$

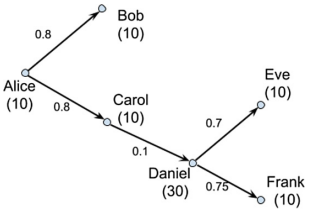
with the graph instance represented as follows:

– 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 guarantees that the person buys the product. This assumption can be removed easily by changing the first rule to a soft rule.

The MEU solution of DT-LP$^{\text{MLN}}$ program $(\Pi^{\text{market}}, \{marketTo(v) \mid v \in V(G)\})$ 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.



4 $p\mathcal{BC}+$ with Utility

We extend $p\mathcal{BC}+$ by introducing the following expression called *utility law* that assigns a reward to transitions:

$$\textbf{reward } v \textbf{ if } F \textbf{ after } G \tag{5}$$

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 (1). We turn a utility law of the form (5) into the LP^{MLN} rule

$$\alpha : \texttt{utility}(v, i+1, id) \leftarrow (i+1 : F) \wedge (i : G) \tag{6}$$

where id is a unique number assigned to the LP^{MLN} rule and $i \in \{0, \ldots, m-1\}$.

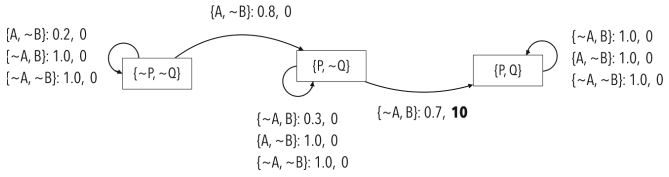
Given a nonnegative integer m denoting the maximum timestamp, a $p\mathcal{BC}+$ action description D with utility over multi-valued propositional signature σ is defined as a high-level representation of the DT-LP^{MLN} program $(Tr(D, m), \sigma_m^{act})$.

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 2. 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 $Pf_1 = \{\text{TRUE} : 0.8, \text{FALSE} : 0.2\}$
caused $Pf_2 = \{\text{TRUE} : 0.7, \text{FALSE} : 0.3\}$
reward 10 **if** $P \wedge Q$ **after** $\neg(P \wedge Q)$
caused $InitP = \{\text{TRUE} : 0.6, \text{FALSE} : 0.4\}$
initially $P = x$ **if** $InitP = x$
caused $InitQ = \{\text{TRUE} : 0.5, \text{FALSE} : 0.5\}$
initially Q **if** $InitQ \wedge P$
initially $\sim Q$ **if** $\sim P$.

The transition system $T(D^{simple})$ *is as follows*:



4.1 Policy Optimization

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 $\pi : \mathbf{S} \times \{0, \ldots, m-1\} \mapsto \mathbf{A}$ that maps a state and a time step to an action (including doing nothing). By $\langle s_0, s_1 \ldots, s_m \rangle^t$ (each $s_i \in \mathbf{S}$) we denote the formula $0\!:\!s_0 \wedge 1\!:\!s_1 \wedge \cdots \wedge m\!:\!s_m$, and by $\langle s_0, a_0, s_1 \ldots, s_{m-1}, a_{m-1}, s_m \rangle^t$ (each $s_i \in \mathbf{S}$ and each $a_i \in \mathbf{A}$) the formula

$$0\!:\!s_0 \wedge 0\!:\!a_0 \wedge 1\!:\!s_1 \wedge \cdots \wedge m-1\!:\!a_{m-1} \wedge m\!:\!s_m.$$

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 from the initial state s_0 is to find a policy π that maximizes the expected utility starting from s_0, i.e., π with

$$\underset{\pi \text{ is a policy}}{\text{argmax}} \; E[U_{Tr(\Pi,m)}(C_{\pi,m} \cup \langle s_0 \rangle^t)]$$

where $C_{\pi,m}$ is the following formula representing policy π:

$$\bigwedge_{s \in \mathbf{S},\ \pi(s,i)=a,\ i \in \{0,\ldots,m-1\}} i\!:\!s \rightarrow i\!:\!a.$$

We define the *total reward* of a history $\langle s_0, a_0, s_1, \ldots, s_m \rangle$ *under the action description* D as

$$R_D(\langle s_0, a_0, s_1, \ldots, s_m \rangle) = E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, a_1, \ldots, a_{m-1}, s_m \rangle^t)].$$

Although it is defined as an expectation, the following proposition tells us that any stable model X of $Tr(D,m)$ such that $X \models \langle s_0, a_0, s_1, \ldots, s_m\rangle$ has the same utility, and consequently, the expected utility of $\langle s_0, a_0, s_1, \ldots, s_m\rangle$ is the same as the utility of any single stable model that satisfies the history.

Proposition 1. *For any two stable models X_1, X_2 of $Tr(D,m)$ that satisfy a history $\langle s_0, a_0, s_1, a_1, \ldots, a_{m-1}, s_m\rangle$, we have*

$$U_{Tr(D,m)}(X_1) \;=\; U_{Tr(D,m)}(X_2) \;=\; E[U_{Tr(D,m)}(\langle s_0, a_0, s_1, a_1, \ldots, a_{m-1}, s_m\rangle^t)].$$

It can be seen that the expected utility of π can be computed from the expected utility from all possible state sequences.

Proposition 2. *Given any initial state s_0 that is consistent with D_{init}, for any non-stationary policy π, we have*

$$E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0\rangle^t)] = \sum_{\langle s_1,\ldots,s_m\rangle : s_i \in \mathbf{S}} R_D(\langle s_0, \pi(s_0), s_1, \ldots, \pi(s_{m-1}), s_m\rangle) \times P_{Tr(D,m)}(\langle s_0, s_1, \ldots, s_m\rangle^t \mid \langle s_0\rangle^t \wedge C_{\pi,m}).$$

Definition 1. *For a $p\mathcal{BC}+$ action description D, let $M(D)$ be the MDP $\langle S, A, T, R\rangle$ where (i) the state set S is $\mathbf{S}$; (ii) the action set A is $\mathbf{A}$; (iii) transition probability T is defined as $T(s,a,s') = P_{D_1}(1:s' \mid 0:s \wedge 0:a)$; (iv) 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 the 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 1. *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 $\langle s_0, s_1, \ldots, s_{m-1}, s_m\rangle$ such that each s_i in $\mathbf{S}$ $(i \in \{0, \ldots, m\})$, we have*

- $R_D(\langle s_0, \pi(s_0), s_1, \ldots, \pi(s_{m-1}), s_m\rangle) = R_{M(D)}(\langle s_0, \pi(s_0,) \ldots, \pi(s_{m-1}), s_m\rangle)$
- $P_{Tr(D,m)}(\langle s_0, s_1, \ldots, s_m\rangle^t \mid \langle s_0\rangle^t \wedge C_{\pi,m}) = P_{M(D)}(\langle s_0, \pi(s_0,) \ldots, \pi(s_{m-1}), s_m\rangle)$.

It follows that the policy optimization problem for $p\mathcal{BC}+$ action descriptions coincides with the policy optimization problem for MDP with finite horizon.

Theorem 2. *For any nonnegative integer m and an initial state $s_0 \in \mathbf{S}$ that is consistent with D_{init}, we have*

$$\underset{\pi \text{ is a non-stationary policy}}{argmax} E[U_{Tr(D,m)}(C_{\pi,m} \wedge \langle s_0\rangle^t)] = \underset{\pi \text{ is a non-stationary policy}}{argmax} ER_{M(D)}(\pi, s_0).$$

Theorem 2 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.

5 $p\mathcal{BC}+$ 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^{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 $p\mathcal{BC}+$ 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.

Example 3. **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 above 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* $\{\texttt{R1}, \texttt{R2}\}$, 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:

$$\mathit{MoveTo}(x, r) \textbf{ causes } \mathit{In}(x) = r \textbf{ if } \mathit{Pf\_Move} \wedge \mathit{GoalNotAchieved}$$
$$\textbf{caused } \mathit{Pf\_Move} = \{\text{TRUE} : p, \text{FALSE} : 1 - p\}.$$

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:*

$$\mathit{MoveTo}(x_1, r_2) \textbf{ causes } \sim \mathit{OnTopOf}(x_1, x_2)$$
$$\textbf{if } \mathit{Pf\_Move} \wedge \mathit{In}(x_1) = r_1 \wedge \mathit{OnTopOf}(x_1, x_2) \wedge \mathit{GoalNotAchieved} \quad (r_1 \neq r_2).$$

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:*

$$\mathit{StackOn}(x_1, x_2) \textbf{ causes } \mathit{OnTopOf}(x_1, x_2)$$
$$\textbf{if } \mathit{TopClear}(x_2) \wedge \mathit{At}(x_1) = r \wedge \mathit{At}(x_2) = r \wedge \mathit{GoalNotAchieved}(x_1 \neq x_2).$$

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:

$$StackOn(x_1, x_2) \textbf{ causes } \sim OnTopOf(x_1, x) \textbf{ if } TopClear(x_2) \wedge At(x_1) = r \wedge At(x_2) = r \wedge OnTopOf(x_1, x) \wedge GoalNotAchieved \quad (x_2 \neq x, x_1 \neq x_2).$$

Two different blocks cannot be on top of the same block, and a block cannot be on top of two different blocks:

$$\textbf{constraint } \neg(OnTopOf(x_1, x) \wedge OnTopOf(x_2, x)) \quad (x_1 \neq x_2)$$
$$\textbf{constraint } \neg(OnTopOf(x, x_1) \wedge OnTopOf(x, x_2)) \quad (x_1 \neq x_2).$$

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:

$$\textbf{default } TopClear(x)$$
$$\textbf{caused } \sim TopClear(x) \textbf{ if } OnTopOf(x_1, x).$$

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:

$$\textbf{caused } Above(x_1, x_2) \textbf{ if } OnTopOf(x_1, x_2)$$
$$\textbf{caused } Above(x_1, x_2) \textbf{ if } Above(x_1, x) \wedge Above(x, x_2).$$

One block cannot be above itself; two blocks cannot be above each other:

$$\textbf{caused } \bot \textbf{ if } Above(x_1, x_2) \wedge Above(x_2, x_1).$$

If a block x_1 is above another block x_2, then x_1 has the same location as x_2:

$$\textbf{caused } At(x_1) = r \textbf{ if } Above(x_1, x_2) \wedge At(x_2) = r. \tag{7}$$

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:*

$$\textbf{caused } GoalNotAchieved \textbf{ if } At(x) = r \quad (r \neq \texttt{R2})$$
$$\textbf{default } \sim GoalNotAchieved.$$

$At(x)$ 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** $\bot$ *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 R2*.*

In Example 3, 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 combination of fluents. On the other hand, the static laws in the above action description reduce the number of states to only $(13 + 9) \times 2 = 44$.[2]

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 (7) 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 invoking an MDP solver MDPTOOLBOX.[3] The current system uses LP^{MLN} 1.0 [9] (http://reasoning.eas.asu.edu/lpmln) 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. The current system is not quite scalable because generating exact transition probability and reward matrices requires enumerating all stable models of D_0 and D_1.

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 [18] 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 Sect. 5. RDDL (Relational Dynamic Influence Diagram Language) [13] 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. Compared to PPDDL and RDDL, the advantages of $p\mathcal{BC}+$ are in its simplicity and expressivity originating from the stable model semantics, which allows for elegant representation of recursive definitions, defeasible behaviors, and indirect effects.

Zhang *et al.* [19] adopt ASP and P-Log [2] which respectively produces 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

[2] This can be verified by counting all possible configurations of 3 blocks with 2 locations.

[3] https://pymdptoolbox.readthedocs.io.

and probability distribution over states take into account commonsense knowledge about the domain, and thus improve the quality of a plan and reduce computation needed at the POMDP level. Yang *et al.* [17] 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 stochastic effects of low-level actions into account. Similarly, Sridharan *et al.* [14] 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.* [12], 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 [4], 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 an ASP solver to obtain states and actions for the RL methods to learn transition probabilities and rewards, and updates 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.

7 Conclusion

Our main contributions are as follows.

- We presented a decision-theoretic extension of LP^{MLN}, through which we extended $p\mathcal{BC}+$ with the language constructs for representing rewards of transitions;
- We showed that the semantics of $p\mathcal{BC}+$ can be equivalently defined in terms of the decision-theoretic LP^{MLN} or MDP;
- We presented the system PBCPLUS2MDP, which solves $p\mathcal{BC}+$ policy optimization problems with an MDP solver.

Formally relating action languages and MDP opens up interesting research to explore. Dynamic programming methods in MDP can be utilized to compute

action languages. In turn, action languages may serve as a formal verification tool for MDP as well as a high-level representation language for MDP that describes an MDP instance in a succinct and elaboration tolerant way. As many reinforcement learning tasks use MDP as a modeling language, the work may be related to incorporating symbolic knowledge to reinforcement learning as evidenced by [12,17,19].

DT-LP$^{\text{MLN}}$ may deserve attention on its own for static domains. We are currently working on an implementation that extends LP$^{\text{MLN}}$ system to handle utility. We expect that the system can be a useful tool for verifying properties for MDP.

The theoretical results in this paper limit attention to MDP in the finite horizon case. When the maximum step m is sufficiently large, we may view it as an approximation of the infinite horizon case, in which case, we allow discount factor γ by replacing v in (6) with $\gamma^{i+1}v$. While it appears intuitive to extend the theoretical results in this paper to the infinite case, it requires extending the definition of LP$^{\text{MLN}}$ to allow infinitely many rules, which we leave for future work.

Acknowledgements. We are grateful to the anonymous referees for their useful comments and to Siddharth Srivastava, Zhun Yang, and Yu Zhang for helpful discussions. This work was partially supported by the National Science Foundation under Grant IIS-1815337.

References

1. Babb, J., Lee, J.: Action language $\mathcal{BC}+$. J. Log. Comput. exv062 (2015). https://doi.org/10.1093/logcom/exv062
2. Baral, C., Gelfond, M., Rushton, J.N.: Probabilistic reasoning with answer sets. Theory Pract. Log. Program. **9**(1), 57–144 (2009)
3. Bellman, R.: A markovian decision process. Indiana Univ. Math. J. **6**, 679–684 (1957)
4. Ferreira, L.A., Bianchi, R.A.C., Santos, P.E., de Mantaras, R.L.: Answer set programming for non-stationary Markov decision processes. Appl. Intell. **47**(4), 993–1007 (2017)
5. Gelfond, M., Lifschitz, V.: Representing action and change by logic programs. J. Log. Program. **17**, 301–322 (1993)
6. Gelfond, M., Lifschitz, V.: Action languages. Electron. Trans. Artif. Intell. **3**, 195–210 (1998). http://www.ep.liu.se/ea/cis/1998/016/
7. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artif. Intell. **153**(1–2), 49–104 (2004)
8. Lee, J., Lifschitz, V., Yang, F.: Action language $\mathcal{BC}+$: preliminary report. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI) (2013)
9. Lee, J., Talsania, S., Wang, Y.: Computing LPMLN using ASP and MLN solvers. Theory Pract. Log. Program. **17**(5–6), 942–960 (2017)
10. Lee, J., Wang, Y.: Weighted rules under the stable model semantics. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 145–154 (2016)

11. Lee, J., Wang, Y.: A probabilistic extension of action language $\mathcal{BC}$+. Theory Pract. Log. Program. **18**(3–4), 607–622 (2018)
12. Leonetti, M., Iocchi, L., Stone, P.: A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. Artif. Intell. **241**, 103–130 (2016)
13. Sanner, S.: Relational dynamic influence diagram language (RDDL): language description. Unpublished ms, p. 32. Australian National University (2010)
14. Sridharan, M., Gelfond, M.: Using knowledge representation and reasoning tools in the design of robots. In: Workshop on Knowledge-Based Techniques for Problem Solving and Reasoning (KnowProS) (2016)
15. Wang, Y., Lee, J.: Elaboration tolerant representation of Markov decision process via decision theoretic extension of pBC+. arXiv e-prints (2019). http://arxiv.org/abs/1904.00512
16. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge, UK, May 1989. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf
17. Yang, F., Lyu, D., Liu, B., Gustafson, S.: PEORL: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In: Proceedings of International Joint Conference on Artificial Intelligence, pp. 4860–4866 (2018)
18. Younes, H.L., Littman, M.L.: PPDDL1.0: an extension to PDDL for expressing planning domains with probabilistic effects. Technical report, CMU-CS-04-162, April 2004
19. Zhang, S., Stone, P.: 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, pp. 1394–1400 (2015)