Bridging Commonsense Reasoning and Probabilistic Planning via a Probabilistic Action Language (Application Paper)

Yi Wang*, Shiqi Zhang#, Joohyung Lee*
**Arizona State University, USA - # SUNY Binghamton, USA*

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

In order to be responsive to dynamically changing real-world environments, an intelligent agent needs to perform complex sequential decision-making tasks that are often guided by commonsense knowledge. The previous work on this line of research led to the framework called interleaved commonsense reasoning and probabilistic planning (iCORPP). iCORPP used P-log to represent commmonsense knowledge and Markov Decision Processes (MDPs) or Partially Observable MDPs (POMDPs) for planning under uncertainty. A main limitation of iCORPP is that its implementation requires non-trivial engineering efforts to bridge the commonsense reasoning and probabilistic planning formalisms. In this paper, we present a unified framework to integrate iCORPP's reasoning and planning components. In particular, we extend probabilistic action language $p\mathcal{BC}+$ to express utility, belief states, and observation as in POMDP models. Inheriting the advantages of action languages, the new action language provides an elaboration tolerant representation of POMDP that reflects commonsense knowledge. The idea led to the design of the system PBCPLUS2POMDP, which compiles a $p\mathcal{BC}+$ action description into a POMDP model that can be directly processed by off-the-shelf POMDP solvers to compute an optimal policy of the $p\mathcal{BC}+$ action description. Our experiments show that it retains the advantages of iCORPP while avoiding the manual efforts in bridging the commonsense reasoner and the probabilistic planner.

1 Introduction

Intelligent agents frequently need to perform complex sequential decision making toward achieving goals that require more than one action, in which the agent's utility depends on a sequence of decisions. A common task is to find the policy that maximizes the agent's utility when the environment is partially observable, i.e., the agent knows only partial information about the current state. Partially Observable Markov Decision Processes (POMDPs) (Kaelbling et al. 1998) have been widely used for that purpose. It assumes partial observability of underlying states and can model the nondeterministic state transitions and local, unreliable observations using probabilities, and plan toward maximizing long-term rewards under such uncertainties. However, as a very general mathematical framework, POMDPs are not equipped with built-in constructs for representing commonsense knowledge.

Recent works (Zhang and Stone 2015; Zhang et al. 2015) aim at embracing commonsense knowledge into probabilistic planning. In that line of research, a reasoner was used for state estimation with contextual knowledge and a planner focuses on selecting actions to maximize the long-term reward. More recently, probabilistic logical knowledge has been used for reasoning about both the current state and the dynamics of the world, resulting in the framework called iCORPP (Zhang et al. 2017). iCORPP builds on two computational paradigms of P-log (Baral et al. 2009) and POMDPs (Kaelbling et al. 1998) for commonsense reasoning and probabilistic planning respectively. Reflecting the commonsense knowledge, iCORPP significantly reduces the complexity of POMDP planning while enabling robot behaviors to adapt to exogenous changes. One example domain in (Zhang et al. 2017)

demonstrates that the MDP constructed by iCORPP includes only 60 states whereas the naive way of enumerating all combinations of attribute values produces more than 2^{69} states.

Despite the advantages, iCORPP has the limitation that practitioners must spend non-trivial engineering efforts to bridge the gap between P-log and POMDP in its implementations. One reason is that P-log does not have the built-in notions of utility and partially observable states as in POMDP models. Thus, the work on iCORPP acquired the transitions and their probabilities by running a P-log solver, but then the user has to manually add the information about the rewards and the belief states (Zhang et al. 2017).

In this paper, we present a more principled way to integrate the commonsense reasoning and probabilistic planning components in the iCORPP framework, which serves as the main contribution of this paper. We extend probabilistic action language $p\mathcal{BC}+$ (Lee and Wang 2018; Wang and Lee 2019) to support the representation of and reasoning with utility, belief states, and observation as in POMDP models. Inheriting the advantages of action languages, the new action language provides an elaboration tolerant representation of POMDP that is convenient to encode commonsense knowledge and completely shield users from the syntax or algorithms of POMDPs.

The second contribution is on the design of the system PBCPLUS2POMDP, which can dynamically construct POMDP models given an action description in $p\mathcal{BC}+$, and compute action policies using off-the-shelf POMDP solvers. Unlike iCORPP, the semantics of $p\mathcal{BC}+$ and its reasoning system together support the direct generation of planning models, which can be further used for computing action policies using POMDP solvers. Experiments have been conducted using the "tiger" and the "dialog management" benchmarks, and the results show that our new language (and its supporting system) retains the advantages of iCORPP while successfully avoiding the manual efforts in bridging the gap between iCORPP's commonsense reasoning and probabilistic planning components.

The paper is organized as follows. After reviewing $p\mathcal{BC}+$ and POMDP in Section 2, we extend $p\mathcal{BC}+$ and show how it can be used to represent POMDP models in Section 3. In Section 4, we show how we can dynamically generate POMDP models by exploiting the elaboration tolerant representation of $p\mathcal{BC}+$. We present the system PBCPLUS2POMDP in Section 5 and experimental results with the system in Section 6. After discussing the related work in Section 7, we conclude in Section 8.

2 Preliminaries

Due to the space limit, the review is brief. We refer the reader to (Lee and Wang 2018; Wang and Lee 2019), or the appendix of this paper for the review of preliminaries.

2.1 Review: $p\mathcal{BC}+$ with Utility

We review $p\mathcal{BC}+$ as presented in (Wang and Lee 2019), which extends the language in (Lee and Wang 2018) by incorporating the concept of utility.

Like its predecessors $\mathcal{BC}$ (Lee et al. 2013) and $\mathcal{BC}$+ (Babb and Lee 2015), 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* is a finite set of *causal laws*, which describes how fluents depend on each other statically

Causal Laws	Syntax	Translation into $\mathrm{LP}^{\mathrm{MLN}}$
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 $j \in \{1,\dots,n\}$: $ln(p_i) : (i{:}c) = v_j$ $(i \in \{0,\dots,m-1\})$
utility law	**reward** v **if** F **after** G where v is a real number, F is a fluent formula, G contains fluent constant and action constant only	$\alpha : \mathtt{utility}(v, i+1, id)$ $\leftarrow (i+1:F) \wedge (i:G)$
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 $\mathrm{LP}^{\mathrm{MLN}}$

and how their values change from one time step to another. Fig. 1 lists causal laws in $p\mathcal{BC}+$ and their translations into $\mathrm{LP}^{\mathrm{MLN}}$ (Lee and Wang 2016). 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 $\mathrm{LP}^{\mathrm{MLN}}$ 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 $\mathrm{LP}^{\mathrm{MLN}}$ 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 $v \in Dom(c)$.

and atoms of the form $\mathtt{utility}(v,i,id)$ introduced by each utility law as described in Fig. 1.

D_m contains $\mathrm{LP}^{\mathrm{MLN}}$ rules obtained from static laws, fluent dynamic laws, utility laws, and pf constant declarations as described in the third column of Fig. 1, as well as $\{0:c=v\}^{\mathrm{ch}}$ for every regular fluent constant c and every $v \in Dom(c)$, and $\{i:c=\textsc{true}\}^{\mathrm{ch}}$, $\{i:c=\textsc{false}\}^{\mathrm{ch}}$ ($i \in \{0,\dots,m-1\}$) 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 $\mathrm{LP}^{\mathrm{MLN}}$ 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 in its domain. We identify an interpretation of σ_m (or σ) that satisfies these constraints with the value assignment function mapping each constant to its value.

For any $\mathrm{LP}^{\mathrm{MLN}}$ 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.

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

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 concurrent execution of actions**: 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.

A $p\mathcal{BC}+$ action description defines 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 $p = P_{D_1}(1\!:\!s' \mid 0\!:\!s \wedge 0\!:\!e)$ and $u = E[U_{D_1}(0\!:\!s \wedge 0\!:\!e \wedge 1\!:\!s')]$. [2] 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 $u(s, e, s')$. The notion of a probabilistic transition system is essentially the same as that of a Markov Decision Process.

2.2 Review: POMDP

A Partially Observable Markov Decision Processes (POMDP) is defined as a tuple

$$\langle S, A, T, R, \Omega, O, \gamma \rangle$$

where (i) S is a set of states; (ii) A is a set of actions; (iii) $T : S \times A \times S \to [0, 1]$ are transition probabilities; (vi) $R : S \times A \times S \to \mathbb{R}$ are rewards; (v) Ω is a set of observations; (vi) $O : S \times A \times \Omega \to [0, 1]$ are observation probabilities; (vii) $\gamma \in [0, 1]$ is a discount factor.

A *belief state* is a probability distribution over S. Given the current belief state b, after taking action $a \in A$ and observing $o \in \Omega$, the updated belief state, b', can be computed as

$$b'(s') = \eta \cdot O(o \mid s', a) \sum_{s \in S} T(s' \mid s, a) b(s)$$

where $s \in S$ and $s' \in S$ are the current and next states respectively; $b(s)$ is the belief probability in b corresponding to s; $b'(s')$ is the belief probability in b' corresponding to s'; and η is a normalizer.

[2] The *utility* of an interpretation I under DT-LP$^{\text{MLN}}$ program Π (Wang and Lee 2019) is defined as $U_\Pi(I) = \Sigma_{\texttt{utility}(u,\texttt{t}) \in I}\, u$ and the *expected utility* of a proposition A is defined as $E[U_\Pi(A)] = \sum_{I \models A} U_\Pi(I) \times P_\Pi(I \mid A)$.

A *policy* π is a function from the set of belief states to the set of actions. The *expected total reward* of a stationary policy π starting from the initial belief state b_0 is

$$V^\pi(b_0) \quad = \sum_{t=0}^{\infty} \gamma^t E\Big[R(s_t, \pi(b_t), s_{t+1}) \mid b_0\Big]$$

where b_t and s_t are the belief state and the state at time t. The optimal policy π^* is obtained by optimizing the long-term reward: $\pi^* = \operatorname*{argmax}_\pi V^\pi(b_0)$.

3 Representing POMDP by Extended $p\mathcal{BC}+$

To be able to express partially observable states, we extend $p\mathcal{BC}+$ by introducing a new type of constants, called *observation constants*, and a new kind of causal laws called *observation dynamic laws*. An *observation dynamic law* is of the form

$$\textbf{observed } F \textbf{ if } G \textbf{ after } H \tag{1}$$

where F is a formula containing no other constants than observation constants, G is a formula containing no other constants than fluent constants, and H is a formula containing no other constants than action constants and pf constants. Observation constants can occur only in observation dynamic laws. An observation dynamic law r of the form (1) is translated into the following $\mathrm{LP}^{\mathrm{MLN}}$ rule:

$$\alpha \ : \ (i+1{:}F) \leftarrow (i+1{:}G) \wedge (i{:}H).$$

For each observation constant obs, $Dom(obs)$ contains a special value `NA` ("Not Applicable"). For each observation constant obs in σ^{obs} and $v \in Dom(obs)$, we include the following $\mathrm{LP}^{\mathrm{MLN}}$ rule in D_m to indicate that the initial value of each observation constant is exogenous:

$$\alpha \ : \ \{0 : obs{=}v\}^{\mathrm{ch}}$$

and include the following $\mathrm{LP}^{\mathrm{MLN}}$ rule in D_m to indicate that the default value of obs is `NA`:

$$\alpha \ : \ \{i : obs{=}\texttt{NA}\}^{\mathrm{ch}} \quad (i \in \{1, \dots, m\}).$$

For a more flexible representation, we introduce the **if** clause in the pf constant declarations as

$$\textbf{caused } c = \{v_1 : p_1, \dots, v_n : p_n\} \textbf{ if } F \tag{2}$$

where c is a pf constant with the domain $\{v_1, \dots, v_n\}$, $0 < p_i < 1$ for each $i \in \{1, \dots, n\}$, $\sum_{i \in \{1,\dots,n\}} p_i = 1$ and F contains rigid constants only.[3] A pf constant declaration (2) is translated into $\mathrm{LP}^{\mathrm{MLN}}$ rules

$$ln(p_i) : (i : c) = v_j \leftarrow F \tag{3}$$

for $j \in \{0, \dots, m\}$. In addition to Assumptions 1–3 above, we add the following assumption:

4. **Rigid constants take same value over all stable models**: for any rigid constant c, there exists $v \in Dom(c)$ such that $I \models c = v$ for all stable model I of D_m.

Under this assumption, the body F in (3) evaluates to either TRUE or FALSE for all stable models of D_m, meaning that either (3) can be removed from D_m, or F can be removed from the body of (3). Thus, this is not an essential extension but helps us use different probability distributions by changing the condition F.

[3] A *rigid* constant is a statically determined fluent constant for which the value is assumed not to change over time (Giunchiglia et al. 2004).

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.

Definition 1
A $p\mathcal{BC}+$ action description D, together with a discount factor γ, defines a POMDP $M(D)$ $\langle S, A, P, R, \Omega, O, \gamma\rangle$ where

- the state set S is the same as $\mathbf{S}$ and the action set A is the same as $\mathbf{A}$;
- the transition probability P is defined as $P(s,a,s') = P_{D_1}(1{:}s' \mid 0{:}s, 0{:}a)$;
- the reward function R is defined as $R(s,a,s') = E[U_{D_1}(0{:}s, 0{:}a, 1{:}s')]$;
- the observation set Ω is the set of interpretations o on σ^{obs} such that $0:o$ is a residual stable model of D_0;
- the observation probability O is defined as $O(s,a,o) = P_{D_1}(1{:}o \mid 1{:}s, 0{:}a)$.

4 Elaboration Tolerant Representation of POMDP

Consider the "dialog management" example from (Zhang et al. 2017): a delivery robot is responsible for delivering an item i to person p in room r. The robot needs to ask questions to figure out what i, p, r are. The challenge comes from the robot's imperfect speech recognition capability. As a result, repeating questions is sometimes necessary. We use POMDPs to model the unreliability from speech recognition, and the robot uses observations to maintain a belief state in the form of a probability distribution. There are two types of questions that the robot can ask:

- Which-Questions: questions about which item/person/room it is, for example, "which item is it?"
- Confirmation-Questions: questions to confirm whether a(n) item/person/room is the requested one, for example, "is the requested item coffee?"

Each of the question-asking action has a small cost. The robot can execute a `deliver` action, which consists of an item i', person p' and room r' as arguments. A `deliver` action deterministically leads to the terminal state. A reward is obtained with `deliver` action, determined by to what extent i', p' and r' matches i, p and r. For instance, when all three entries are correctly identified in the `deliver` action, the agent receives a large reward; when none is correctly identified, the agent receives a large penalty (in the form of a negative reward). Therefore, the agent has the motivation of computing action policies to minimize the cost of its question-asking actions, while maximizing the expected reward by tasking the "correct" delivery action.

This example can be represented in $p\mathcal{BC}+$ as follows. We assume a small domain where $\mathit{Item} = \{\mathit{Coffee}, \mathit{Coke}, \mathit{Cookies}, \mathit{Burger}\}$, $\mathit{Person} = \{\mathit{Alice}, \mathit{Bob}, \mathit{Carol}\}$, $\mathit{Room} = \{R_1, R_2, R_3\}$.

Notation: i, i' range over *Item*, p, p' ranges over *Person*, r, r' ranges over *Room*, c ranges over $\{\texttt{Yes}, \texttt{No}\}$

Observation constant:	Domains:
ItemObs	*Item* $\cup \{\texttt{NA}\}$
PersonObs	*Person* $\cup \{\texttt{NA}\}$
RoomObs	*Room* $\cup \{\texttt{NA}\}$
Confirmed	$\{\texttt{Yes}, \texttt{No}, \texttt{NA}\}$
Regular fluent constants:	Domains:
ItemReq	*Item*

PersonReq	*Person*
RoomReq	*Room*
Terminated	Boolean
Action constants:	Domains:
WhichItem, *WhichPerson*, *WhichRoom*, *ConfirmItem*(i), *ConfirmPerson*(p), *ConfirmRoom*(r), *Deliver*(i,p,r)	Boolean
Pf constants:	Domains:
Pf_WhichItem(i)	*Item*
Pf_WhichPerson(p)	*Person*
Pf_WhichRoom(r)	*Room*
Pf_ConfirmWhenCorrect, *Pf_ConfirmWhenIncorrect*	$\{\texttt{Yes}, \texttt{No}\}$

The action *Deliver* causes the entering of the terminal state:

$$\textbf{caused } \textit{Terminated} \textbf{ if } \top \textbf{after } \textit{Deliver}(i,p,r).$$

The execution of *Deliver* action with the room, the person and the item all correct yields a reward of r. The execution of *Deliver* action with a wrong item, a wrong person, or a wrong room yield a penalty of p_1, p_2, p_3 each.

$$\begin{aligned}
&\textbf{reward } r \textbf{ if } \textit{ItemReq} = i \wedge \textit{PersonReq} = p \wedge \textit{RoomReq} = r \wedge \textit{Deliver}(i,p,r) \wedge \sim\textit{Terminated},\\
&\textbf{reward } -p_1 \textbf{ if } \textit{ItemReq} = i \wedge \textit{Deliver}(i',p',r') \wedge \sim\textit{Terminated} \quad (i \neq i'),\\
&\textbf{reward } -p_2 \textbf{ if } \textit{PersonReq} = p \wedge \textit{Deliver}(i',p',r') \wedge \sim\textit{Terminated} \quad (p \neq p'),\\
&\textbf{reward } -p_3 \textbf{ if } \textit{RoomReq} = r \wedge \textit{Deliver}(i',p',r') \wedge \sim\textit{Terminated} \quad (r \neq r').
\end{aligned}$$

Asking "which item" question when the actual item being requested is i returns an item i' as observation in accordance with the probability distribution defined by pf constant *Pf_WhichItem*(i), shown below. "Which person" and "Which room" questions are represented in a similar way.

$$\begin{aligned}
&\textbf{observed } \textit{ItemObs} = i' \textbf{ if } \textit{ItemReq} = i \wedge \sim\textit{Terminated} \textbf{ after } \textit{WhichItem} \wedge \textit{Pf_WhichItem}(i) = i',\\
&\textbf{caused } \textit{Pf_WhichItem}(\textit{Coffee}) = \{\textit{Coffee} : 0.7, \textit{Coke} : 0.1, \textit{Cookies} : 0.1, \textit{Burger} : 0.1\},\\
&\textbf{caused } \textit{Pf_WhichItem}(\textit{Coke}) = \{\textit{Coffee} : 0.1, \textit{Coke} : 0.7, \textit{Cookies} : 0.1, \textit{Burger} : 0.1\},\\
&\textbf{caused } \textit{Pf_WhichItem}(\textit{Cookies}) = \{\textit{Coffee} : 0.1, \textit{Coke} : 0.1, \textit{Cookies} : 0.7, \textit{Burger} : 0.1\},\\
&\textbf{caused } \textit{Pf_WhichItem}(\textit{Burger}) = \{\textit{Coffee} : 0.1, \textit{Coke} : 0.1, \textit{Cookies} : 0.1, \textit{Burger} : 0.7\},
\end{aligned} \tag{4}$$

When the robot asks the confirmation question "is the item i?", the human's answer could be sometimes mistakenly recognized, and the probability distribution of the answer depends on whether the item i is indeed what the human asked for. We use two pf constants, *Pf_ConfirmWhenCorrect* and *Pf_ConfirmWhenIncorrect* to specify the distinct probability distributions depending on whether the robot's guess is correct or not. When the robot asks to confirm if the item requested is i, which is indeed what the human requested:

$$\begin{aligned}
&\textbf{observed } \textit{Confirmation} = v \textbf{ if } \textit{ItemReq} = i \wedge \sim\textit{Terminated}\\
&\qquad\qquad \textbf{after } \textit{ConfirmItem}(i) \wedge \textit{Pf_ConfirmWhenCorrect} = v. \qquad (v \in \{\texttt{Yes}, \texttt{No}\})\\
&\textbf{caused } \textit{Pf_ConfirmWhenCorrect} = \{\texttt{Yes} : 0.8, \texttt{No} : 0.2\}.
\end{aligned}$$

When the robot asks to confirm if the requested item is i' whereas the actual item the human requested

is i':

observed *Confirmation*$=v$ **if** *ItemReq*$=i\wedge\sim$*Terminated*
after *ConfirmItem*$(i')\wedge$ *Pf_ConfirmWhenIncorrect*$=v \qquad (i\neq i')$,
caused *Pf_ConfirmWhenIncorrect*$=\{\texttt{Yes}:0.2,\texttt{No}:0.8\}$.

(The probability distributions of these pf constants do not have to be complementary.)

The formulations of person- and room-related questions are defined similarly, and omitted from the paper.

Asking which-questions has a cost of c_1; asking confirmation-questions has a cost of c_2.

reward c_1 **if** $\top$ **after** *WhichItem*, **reward** c_2 **if** $\top$ **after** *ConfirmItem*(i),
reward c_1 **if** $\top$ **after** *WhichPerson*, **reward** c_2 **if** $\top$ **after** *ConfirmPerson*(p),
reward c_1 **if** $\top$ **after** *WhichRoom*, **reward** c_2 **if** $\top$ **after** *ConfirmRoom*(r).

Finally, all regular fluents in this domain are inertial:

inertial *rf* $\qquad$ (*rf* $\in\{$*ItemReq*, *PersonReq*, *RoomReq*, *Terminated*$\}$).

We illustrate that the above $p\mathcal{BC}+$ action description is elaboration tolerant through the following elaborations. It should be noted that using vanilla POMDP methods, manipulating states, actions, or observation functions requires a lot of engineering efforts, and people frequently have to tune prohibitively a large number of parameters. iCORPP and this research aim to avoid that through probabilistic reasoning about actions. In this work, we move forward from iCORPP to completely shield developers from the syntax or algorithms of POMDPs.

4.1 Elaboration 1: Unavailable items

When an item becomes unavailable for delivery, we can simply remove that item from the domains of relevant constants. For example, when *Coke* becomes unavailable, we simply replace the pf constant declarations in (4) with

caused *Pf_WhichItem*(*Coffee*)$=\{$*Coffee* $:0.78$, *Cookies* $:0.11$, *Burger* $:0.11\}$,
caused *Pf_WhichItem*(*Cookies*)$=\{$*Coffee* $:0.11$, *Cookies* $:0.78$, *Burger* $:0.11\}$,
caused *Pf_WhichItem*(*Burger*)$=\{$*Coffee* $:0.11$, *Cookies* $:0.11$, *Burger* $:0.78\}$.

4.2 Elaboration 2: Reflecting personal preference in reward function

We use a rigid fluent *Interchangeable*(p,i_1,i_2) with the integer domain to represent to what degree the two items i_1,i_2 are interchangeable for person p. For example, Alice does not mind when the robot delivers coke while she actually ordered coffee but she does mind when the robot delivers burger instead of coffee. We add the following elaboration to represent object interchangeabilities.

caused *Interchangeable*(*Alice*, *Coffee*, *Coke*)$=5$,
caused *Interchangeable*(*Alice*, *Coffee*, *Cookies*)$=1$,
caused *Interchangeable*(*Alice*, *Coffee*, *Burger*)$=-3$.

We add the following causal law to reflect the interchageability of the items.

reward x **if** *ItemReq*$=i\wedge$ *Interchangeable*$(p,i,i')=x\wedge$ *PersonReq*(p) **after** *Deliver*(i',p',r').

Such knowledge can be used to enable the robot to be more conservative in delivering items, such as *burger*, due to their low interchangeability to other items.

4.3 Elaboration 3: Changing Perception Model

The speech recognition system may have different accuracy depending on the environment. For example, when there is background noise, its accuracy could drop. In this case, we can update the probability distribution for the relevant pf constant, controlled by auxiliary constants indicating the situation. We introduce a rigid constant called *Noise*, then we replace (4) with

$$\begin{aligned}
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Coffee}) = \{\mathit{Coffee} : 0.7, \mathit{Coke} : 0.1, \mathit{Cookies} : 0.1, \mathit{Burger} : 0.1\} \textbf{ unless } ab \\
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Coke}) = \{\mathit{Coffee} : 0.1, \mathit{Coke} : 0.7, \mathit{Cookies} : 0.1, \mathit{Burger} : 0.1\} \textbf{ unless } ab \\
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Cookies}) = \{\mathit{Coffee} : 0.1, \mathit{Coke} : 0.1, \mathit{Cookies} : 0.7, \mathit{Burger} : 0.1\} \textbf{ unless } ab \\
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Burger}) = \{\mathit{Coffee} : 0.1, \mathit{Coke} : 0.1, \mathit{Cookies} : 0.1, \mathit{Burger} : 0.7\} \textbf{ unless } ab
\end{aligned} \tag{5}$$

to make them defeasible. We then define the probability distribution to override the original ones when there is loud background noise.

$$\begin{aligned}
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Coffee}) = \{\mathit{Coffee} : \frac{6}{10}, \mathit{Coke} : \frac{4}{30}, \mathit{Cookies} : \frac{4}{30}, \mathit{Burger} : \frac{4}{30}\} \textbf{ if } \mathit{Noise}, \\
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Coke}) = \{\mathit{Coffee} : \frac{4}{30}, \mathit{Coke} : \frac{6}{10}, \mathit{Cookies} : \frac{4}{30}, \mathit{Burger} : \frac{4}{30}\} \textbf{ if } \mathit{Noise}, \\
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Cookies}) = \{\mathit{Coffee} : \frac{4}{30}, \mathit{Coke} : \frac{4}{30}, \mathit{Cookies} : \frac{6}{10}, \mathit{Burger} : \frac{4}{30}\} \textbf{ if } \mathit{Noise}, \\
&\textbf{caused } \mathit{Pf_WhichItem}(\mathit{Burger}) = \{\mathit{Coffee} : \frac{4}{30}, \mathit{Coke} : \frac{4}{30}, \mathit{Cookies} : \frac{4}{30}, \mathit{Burger} : \frac{6}{10}\} \textbf{ if } \mathit{Noise}.
\end{aligned}$$

We add

$$\textbf{caused } ab \textbf{ if } \mathit{Noise}$$

to indicate that by default there is no background noise. When the robot agent detects that there is background noise, we add

$$\textbf{caused } \mathit{Noise}$$

to the action description to update the generated POMDP to incorporate the new speech recognition probabilities. It should be noted that the speech recognition component is generally unreliable, though background noise further reduces its reliability.

5 System PBCPLUS2POMDP

We implemented the prototype system PBCPLUS2POMDP, which takes a $p\mathcal{BC}+$ action description D as input and outputs the POMDP $M(D)$ in the input language of the POMDP solver APPL [4]. The system uses LPMLN2ASP (Lee et al. 2017) with exact inference on D_1 and D_0 to generate the components of POMDP: all states, all actions, all transitions and their probabilities, all observations and their probabilities and transition rewards as defined in Definition 1. The system is publicly available at `https://github.com/ywang485/pbcplus2pomdp`, along with several examples.

Even though we limit the computation to D_0 and D_1, i.e., at most one step action execution is considered, the number of stable models may become too huge to enumerate all. Since the transition probabilities, rewards, observation probabilities are per each action, the system implements a compositional way to generate the POMDP model by partitioning the actions in different groups and

[4] `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`

	POMDP Generation Time		POMDP Solving Time (APPL)		
Domain Size	PBCPLUS2POMDP (naive)	PBCPLUS2POMDP (compo)	$\gamma = 0.9$	$\gamma = 0.8$	$\gamma = 0.7$
$2i2p2r$ $\#states = 16$ $\#actions = 18$ $\#observations = 9$	49m10.495s	0m13.611s	0m6.123s	0m0.680s	0m0.249s
$2i3p2r$ $\#states = 24$ $\#actions = 23$ $\#observations = 10$	$>$ 1hr	0m22.723s	4m43.572s	0m21.939s	0m2.294s
$3i3p2r$ $\#states = 36$ $\#actions = 30$ $\#observations = 11$	$>$ 1hr	0m41.944s	$>$ 1hr	8m14.415s	0m37.944s
$4i3p2r$ $\#states = 48$ $\#actions = 37$ $\#observations = 12$	$>$ 1hr	2m56.652s	$>$ 1hr	$>$ 1hr	10m50.248s

Table 1. *Running Statistics of POMDP Model Generation and Solving in Dialog Example*

generating the POMDP model per each group by omitting the causal laws involving other actions and their pf constants. This "compositional" mode often saves the POMDP generation time drastically.[5]

6 Evaluation

All experiments reported in this section were performed on a machine powered by 4 Intel(R) Core(TM) i5-2400 CPU with OS Ubuntu 14.04.5 LTS and 8G memory.

6.1 Evaluation of Planning Efficiency

We report the running statistics of POMDP generation with our PBCPLUS2POMDP system and POMDP planning with APPL on the dialog example (as described in Section 4) in Table 1. We test domains with different numbers of items, people and rooms. PBCPLUS2POMDP(NAIVE) generates POMDP in a non-compositional way while PBCPLUS2POMDP(COMPO) generates POMDP in a compositional way (as described in Section 5) by partitioning actions into $\{\mathit{ConfirmItem}(i) \mid i \in \mathit{Item}\}$, $\{\mathit{ConfirmPerson}(p) \mid p \in \mathit{Person}\}$, $\{\mathit{ConfirmRoom}(r) \mid r \in \mathit{Room}\}$, $\{\mathit{WhichItem}\}$, $\{\mathit{WhichPerson}\}$, $\{\mathit{WhichRoom}\}$, $\{\mathit{Deliver}(i,p,r) \mid i \in \mathit{Item}, p \in \mathit{Person}, r \in \mathit{Room}\}$.

γ is a discount factor. "POMDP solving time (APPL)" refers to the running time of APPL until the convergence to a target precision of 0.1. The PBCPLUS2POMDP(COMPO) mode is much more efficient than the PBCPLUS2POMDP(NAIVE) mode for the dialog domain.

6.2 Evaluation of Solution Quality

$p\mathcal{BC}+$ provides a high-level description of POMDP models such that various elaborations on the underlying action domain can be easily achieved by changing a small part of the $p\mathcal{BC}+$ action description, whereas such elaboration would require a complete reconstruction of transition/reward/observation matrices at POMDP level. In Sections 4.1, 4.2 and 4.3, we have illustrated this point with

[5] The more detailed description of the algorithm is given in **??**.

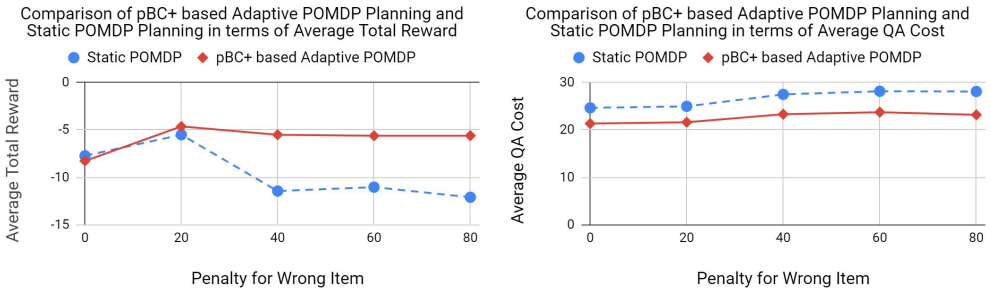

Fig. 2. Impact of Elaboration 1 on Policy Generated

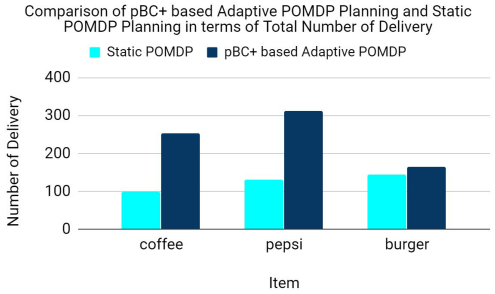

Fig. 3. Impact of Elaboration 2 on Policy Generated

the three example elaborations. In this subsection, we evaluate the impact of the three elaborations on dynamic planning, in the sense that the low-level POMDP (planning module) can be updated automatically once the high-level $p\mathcal{BC}+$ action description (reasoning module) detects changes in the environment to generate better plans. For each of the thee elaborations, we compare the plan generated from a static POMDP that does not reflect environmental changes, and the one generated from the adaptive POMDP that is updated by $p\mathcal{BC}+$ reasoning to reflect environmental changes.

Fig. 2 compares the policies generated from the static POMDPs (baseline) and from the POMDP dynamically generated using $p\mathcal{BC}+$, where the two items of burger and cookies might be unavailable (Elaboration 1). We have run 1000 simulation trials. The diagram on the left compares them in terms of average total reward from the simulation runs, and the right is in terms of average QA cost (accumulated penalty by asking questions). In this experiment, the discount factor is 0.95 (which offers the dialog agent a relatively long horizon), c_1 is 4.0, c_2 is 2.0, r is 20.0, p_2 is 20.0, and p_3 is 30.0. Action policies are generated using APPL in at most 120 seconds. We observe that the adaptive POMDP (ours) achieves a higher average total reward when the penalty for the wrong item is positive, and the adaptive POMDPs are able to complete deliveries with less QA cost. It is worth noting that by reflecting unavailable items, $p\mathcal{BC}+$ reduces the size of the generated POMDP models, resulting in shorter POMDP-solving time. As can be seen from Table 1, for a domain that contains 2 items, 3 people and 2 rooms, POMDP generation plus POMDP solving takes way less time than POMDP solving on a domain with 4 items, 3 people and 2 rooms.

Fig. 3 compares the policies generated from the static POMDP and from $p\mathcal{BC}+$ based adaptive POMDP when item interchangeability is introduced (Elaboration 2). We replaced cookies with pepsi in the domain, added causal laws to indicate that when coke is being requested, delivering pepsi yields a reward of 15, delivering coffee yields a reward of 5 and delivering burger yields an additional penalty of 20 (in the presence of penalty p_1). We have run 10000 simulations, and for all of the simulations, the actual item being requested is fixed to be coke.[6] For the static POMDP, 9628 deliv-

[6] The item is fixed to be coke only during simulation, not during policy generation.

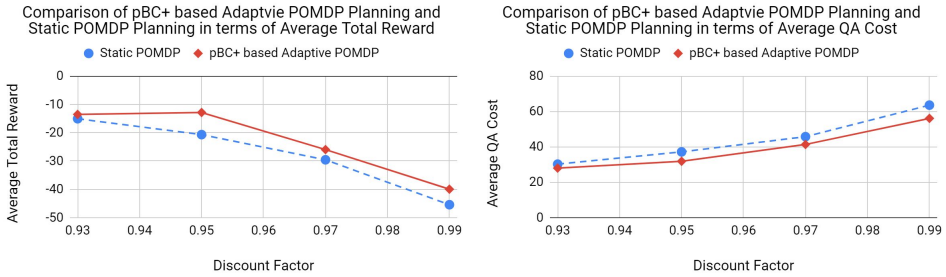

Fig. 4. Impact of Elaboration 3 on Policy Generated

eries were correct, and for the adaptive POMDP, 9270 deliveries were correct. Note that although the static POMDP achieves more correct deliveries, the dynamically generated POMDPs (our approach) achieved higher average total reward by asking fewer questions. The policy generated from the static POMDP gives similar numbers of deliveries for each item that is not coke, while the policy generated from the adaptive POMDP delivered pepsi the most and burger the least, which is aligned with our setting of interchangeability. The discount factor for this experiment is set to be 0.99. c_1 is 6, c_2 is 4, r is 5, p_1 is 5, p_2 is 20 and p_3 is 30. Policies from both POMDPs are generated by APPL with 120 seconds.

Fig. 4 compares the policies generated from the static POMDP and from $p\mathcal{BC}+$ based adaptive POMDP when there is a background noise (Elaboration 3). To reflect environmental noise, we lowered the observation probability of correct answers by 0.1 (and the remaining answers are uniformly distributed). We have run 1000 simulations. The diagram on the left compares them in term of average total reward from the simulation runs, and the diagram on the right compares them in terms of average QA cost (accumulated cost from questions asked) from the simulation runs. In this experiment, c_1 is 4, c_2 is 2, r is 20, p_2 is 20 and p_3 is 30. Policies from both POMDPs are generated by APPL with 120 seconds. It can be seen from the diagrams that while the average total reward of both POMDPs decreases as the discount factor increases, the adaptive POMDP achieves higher average total reward by asking fewer questions.

7 Related Work

Intelligent agents need the capabilities of both reasoning about declarative knowledge, and probabilistic planning toward achieving long-term goals. A variety of algorithms have been developed to integrate commonsense reasoning and probabilistic planning (Hanheide et al. 2017; Zhang et al. 2015; Zhang and Stone 2015; Sridharan et al. 2019; Chitnis et al. 2018; Zhang et al. 2017; Amiri et al. 2018), and some of them, such as (Sridharan et al. 2019) and (Amiri et al. 2018), also include non-deterministic dynamic laws for observations. Although the algorithms use very different computational paradigms for representing and reasoning with human knowledge (e.g., logics, probabilities, graphs, etc), they all share the goal of leveraging declarative knowledge to improve the performance in probabilistic planning. In these works, the hypothesis is that human knowledge potentially can be useful in guiding robot behaviors in the real world, while the challenge is that human knowledge is sparse, incomplete, and sometimes unreliable. In this research, we share the same goal of utilizing contextual knowledge from people to help intelligent agents in sequential decision-making tasks while accounting for the uncertainty in perception and action outcomes.

Among the algorithms that integrate commonsense reasoning and probabilistic planning paradigms, iCORPP enabled an agent to reason with contextual knowledge to dynamically construct complete probabilistic planning models (Zhang et al. 2017) for adaptive robot control, where P-log was used

for logical-probabilistic reasoning (Baral et al. 2009). Depending on the observability of world states, iCORPP uses either Markov Decision Processes (MDPs) (Puterman 2014) or Partially Observable MDPs (POMDPs) (Kaelbling et al. 1998) for probabilistic planning. As a result, iCORPP has been applied to robot navigation, dialog system, and manipulation tasks (Zhang et al. 2017; Amiri et al. 2018). In this work, we develop a unified representation and a corresponding implementation for iCORPP, where the entire reasoning and planning system can be encoded using a single program, and practitioners are completely shielded from the technical details of formulating and solving (PO)MDPs. In comparison, iCORPP requires significant engineering efforts (e.g., using Python or C++) for "gluing" the computational paradigms used by the commonsense reasoning and probabilistic planning components.

Recently, researchers have developed algorithms to incorporate knowledge representation and reasoning into reinforcement learning (RL) (Sutton and Barto 2018), where the goal is to provide the learning agents with guidance in action selections through reasoning with declarative knowledge. Notable examples include (Leonetti et al. 2016; Yang et al. 2018; Jiang et al. 2018; Lu et al. 2018; Lyu et al. 2019; Kim et al. 2019). In this research, we assume the availability of world models, including both states and dynamics, in a declarative form. In case of world models being unavailable, incomplete, or dynamically changing, there is the potential of combining the above "knowledge-driven RL" algorithms, particularly the ones using model-based RL such as (Lu et al. 2018), with our new representation to enable agents to simultaneously learn and reason about world models to compute action policies.

In an earlier work (Tran and Baral 2004), the authors show how Pearl's probabilistic causal model can be encoded in a probabilistic action language PAL (Baral et al. 2002).

8 Conclusion and Future Work

In this paper, we present a principled way of integrating probabilistic logical reasoning and probabilistic planning. This is done by extending probabilistic action language $p\mathcal{BC}+$ (Lee and Wang 2018; Wang and Lee 2019) to be able to express utility, belief states, and observation as in POMDP models. Inheriting the advantages of action languages, the new action language provides an elaboration tolerant representation of POMDP that is convenient to encode commonsense knowledge.

One of the well known problems limiting applications of POMDPs is sensitivity of the optimal behavior to the small changes in the reward function and the probability distribution. Because of this sensitivity care must be taken in choosing the reward function as well as the probability distribution. The choice of these, and especially of the latter is a non-trivial problem, which is outside of the scope of the paper. POMDP algorithms perform poorly in scalability in many applications. Although the language and system developed in this paper can potentially alleviate this issue, we believe this is a challenging problem that deserves more effort, and we leave it to future work.

The current prototype implementation is not highly scalable when the number of transitions becomes large. For a more scalable generation of the POMDP input using the $\mathrm{LP}^{\mathrm{MLN}}$ system, we could use the sampling method in $\mathrm{LP}^{\mathrm{MLN}}$ inference, which we leave for future work.

References

AMIRI, S., WEI, S., ZHANG, S., SINAPOV, J., THOMASON, J., AND STONE, P. 2018. Multi-modal predicate identification using dynamically learned robot controllers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*.

BABB, J. AND LEE, J. 2015. Action language $\mathcal{BC}+$. *Journal of Logic and Computation*, exv062.

BARAL, C., GELFOND, M., AND RUSHTON, J. N. 2009. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming 9,* 1, 57–144.

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)*, pp. 507–512.

CHITNIS, R., KAELBLING, L. P., AND LOZANO-PÉREZ, T. 2018. Integrating human-provided information into belief state representation using dynamic factorization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3551–3558. IEEE.

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

HANHEIDE, M., GÖBELBECKER, M., HORN, G. S., PRONOBIS, A., SJÖÖ, K., AYDEMIR, A., JENSFELT, P., GRETTON, C., DEARDEN, R., JANICEK, M., ET AL. 2017. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence 247*, 119–150.

JIANG, Y., YANG, F., ZHANG, S., AND STONE, P. 2018. Integrating task-motion planning with reinforcement learning for robust decision making in mobile robots. *CoRR abs/1811.08955*.

KAELBLING, L. P., LITTMAN, M. L., AND CASSANDRA, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence 101,* 1-2, 99–134.

KIM, B., KAELBLING, L. P., AND LOZANO-PEREZ, T. 2019. Adversarial actor-critic method for task and motion planning problems using planning experience. In *AAAI Conference on Artificial Intelligence (AAAI)*.

LEE, J., LIFSCHITZ, V., AND YANG, F. 2013. Action language $\mathcal{BC}$: Preliminary report. 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*, 17(5-6):942-960.

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)*, pp. 145–154.

LEE, J. AND WANG, Y. 2018. A probabilistic extension of action language $\mathcal{BC}$+. *Theory and Practice of Logic Programming 18(3–4)*, 607–622.

LEONETTI, M., IOCCHI, L., AND STONE, P. 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence 241*, 103–130.

LU, K., ZHANG, S., STONE, P., AND CHEN, X. 2018. Robot representing and reasoning with knowledge from reinforcement learning. *CoRR abs/1809.11074*.

LYU, D., YANG, F., LIU, B., AND GUSTAFSON, S. 2019. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *AAAI*.

PUTERMAN, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

SRIDHARAN, M., GELFOND, M., ZHANG, S., AND WYATT, J. 2019. REBA: A refinement-based architecture for knowledge representation and reasoning in robotics. *Journal of Artificial Intelligence Research 65*, 87–180.

SUTTON, R. S. AND BARTO, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

TRAN, N. AND BARAL, C. 2004. Encoding probabilistic causal model in probabilistic action language. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 305–310.

WANG, Y. AND LEE, J. 2019. Elaboration tolerant representation of markov decision process via decision theoretic extension of action language pbc+. In *LPNMR*.

YANG, F., LYU, D., LIU, B., AND GUSTAFSON, S. 2018. Peorl: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 4860–4866.

ZHANG, S., KHANDELWAL, P., AND STONE, P. 2017. Dynamically constructed (PO)MDPs for adaptive robot planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 3855–3863.

ZHANG, S., SRIDHARAN, M., AND WYATT, J. L. 2015. Mixed logical inference and probabilistic planning for robots in unreliable worlds. *IEEE Transactions on Robotics 31,* 3, 699–713.

ZHANG, S. AND STONE, P. 2015. CORPP: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.