PPDDL1.0: The Language for the Probabilistic Part of IPC-4

Håkan L. S. Younes
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
lorens@cs.cmu.edu

Michael L. Littman
Department of Computer Science
Rutgers University
Piscataway, NJ 08854, USA
mlittman@cs.rutgers.edu

Introduction

A standard domain description language, PDDL (Ghallab *et al.* 1998; McDermott 2000; Fox & Long 2003), for deterministic planning domains has simplified sharing of domain models and problems in the planning community, and has enabled direct comparisons of different planning systems. As a result, there has been considerable progress in planning research with deterministic domain models since the first International Planning Competition in 1998.

The 4th International Planning Competition includes a probabilistic track for the first time in an attempt to create a common platform for the evaluation of probabilistic and decision-theoretic planning systems. This document briefly describes the input language, PPDDL1.0, that was used for the probabilistic track. PPDDL1.0 is essentially a syntactic extension of levels 1 and 2 of PDDL2.1 (Fox & Long 2003). We assume that the reader is familiar with PDDL2.1, so focus on the new language features, which include probabilistic effects and rewards. The semantics of a PPDDL1.0 planning problem is given in terms of a Markov decision process (Howard 1960).

Note that, unlike PDDL2.1, we do not impose a specific structure on plans in PPDDL1.0. Planning systems are evaluated using a client-server model in the probabilistic track of the competition. During evaluation of a planner, the server send a state to the client (planning system), which in return sends an action to be executed in the given state. The problem of plan representation is left entirely to the planning systems.

Probabilistic Effects

In order to define probabilistic and decision-theoretic planning problems, we need to add support for probabilistic effects. The syntax for probabilistic effects is

`(probabilistic` p_1 e_1 $\ldots$ p_k e_k`)`

meaning that effect e_i occurs with probability p_i. We require that the constraints $p_i \geq 0$ and $\sum_{i=1}^{k} p_i = 1$ are fulfilled: a probabilistic effect declares an exhaustive set of probability-weighted outcomes. However, we allow a probability-effect pair to be left out if the effect is empty. In other words,

`(probabilistic` p_1 e_1 $\ldots$ p_l e_l`)`

with $\sum_{i=1}^{l} p_i \leq 1$ is syntactic sugar for

Name	Type	Init 1	Init 2
$bomb\text{-}in\text{-}package_{\text{package1}}$	boolean	true	false
$bomb\text{-}in\text{-}package_{\text{package2}}$	boolean	false	true
$toilet\text{-}clogged$	boolean	false	false
$bomb\text{-}defused$	boolean	false	false

Table 1: State variables and their initial values for the "Bomb and Toilet" problem.

`(probabilistic` p_1 e_1 $\ldots$ p_l e_l q `(and))`

with $q = 1 - \sum_{i=1}^{l} p_i$. For example, the effect

```
(probabilistic 0.9 (clogged))
```

means that with probability 0.9 the state variable $clogged$ becomes true in the next state, while with probability 0.1 the state remains unchanged. Outcomes are not required to be mutually exclusive. A new requirements flag is introduced to signal that support for probabilistic effects is required:

```
:probabilistic-effects
```

Figure 1 shows an encoding in PPDDL of the "Bomb and Toilet" example described by Kushmerick, Hanks, & Weld (1995). In this problem, there are two packages, one of which contains a bomb. The bomb can be defused by dunking the package containing the bomb in the toilet. There is a 0.05 probability of the toilet becoming clogged when a package is placed in it. The problem definition in Figure 1 also shows that initial conditions in PPDDL can be probabilistic. In this particular example we define two possible initial states with equal probability (0.5) of being the true initial state. Table 1 lists the state variables for the "Bomb and Toilet" problem and their values in the two possible initial states. Intuitively, we can think of the initial conditions of a PPDDL planning problem as being the effects of an action forced to be scheduled right before time 0. Also, note that the goal of the problem involves negation, which is why the problem definition declares the `:negative-preconditions` requirements flag.

PPDDL allows arbitrary nesting of conditional and probabilistic effects. This is in contrast to popular propositional encodings, such as probabilistic STRIPS operators (PSOs) (Kushmerick, Hanks, & Weld 1995) and factored PSOs (Dearden & Boutilier 1997), which do not allow conditional effects nested inside probabilistic effects. While arbitrary

```
(define (domain bomb-and-toilet)
   (:requirements :conditional-effects :probabilistic-effects)
   (:predicates (bomb-in-package ?pkg) (toilet-clogged) (bomb-defused))
   (:action dunk-package
            :parameters (?pkg)
            :effect (and (when (bomb-in-package ?pkg) (bomb-defused))
                         (probabilistic 0.05 (toilet-clogged)))))

(define (problem bomb-and-toilet)
   (:domain bomb-and-toilet)
   (:requirements :negative-preconditions)
   (:objects package1 package2)
   (:init (probabilistic 0.5 (bomb-in-package package1)
                         0.5 (bomb-in-package package2)))
   (:goal (and (bomb-defused) (not (toilet-clogged)))))
```

Figure 1: PPDDL encoding of "Bomb and Toilet" example.

nesting does not add to the expressiveness of the language, it can allow for exponentially more compact representations of certain effects given the same set of state variables and actions (Rintanen 2003). However, any PPDDL action can be translated into a *set* of PSOs with at most a polynomial increase in size of the representation. Consequently, it follows from the results of Littman (1997) that PPDDL is representationally equivalent to dynamic Bayesian networks (Dean & Kanazawa 1989), which is another popular representation for MDP planning problems.

Rewards and Plan Objectives

Markovian rewards, associated with state transitions, can be encoded using fluents. PPDDL reserves the fluent $reward$, accessed as `(reward)` or `reward`, to represent the total accumulated reward since the start of execution. Rewards are associated with state transitions through update rules in action effects. The use of the $reward$ fluent is restricted to action effects of the form

(⟨*additive-op*⟩ ⟨*reward fluent*⟩ ⟨*f-exp*⟩)

where ⟨*additive-op*⟩ is either `increase` or `decrease`, and ⟨*f-exp*⟩ is a numeric expression not involving $reward$. Action preconditions and effect conditions are not allowed to refer to the $reward$ fluent, which means that the accumulated reward does not have to be considered part of the state space. The initial value of $reward$ is zero. These restrictions on the use of the $reward$ fluent allow a planner to handle domains with rewards, without having to implement full support for fluents.

The requirements flag, `:rewards`, is introduced to signal that support for Markovian rewards is required. Domains that require both probabilistic effects and rewards can declare the `:mdp` requirements flag, which implies `:probabilistic-effects` and `:rewards`.

Figure 2 shows part of the PPDDL encoding of a coffee delivery domain described by Dearden & Boutilier (1997). A reward of 0.8 is awarded if the user has coffee when the "buy-coffee" action is executed, and a reward of 0.2 is awarded when "buy-coffee" is executed in a state where $is\text{-}wet$ is false. Note that a total reward of 1.0 can be awarded as a result of executing the "buy-coffee" action if it is executed in a state where both $user\text{-}has\text{-}coffee$ and $\neg is\text{-}wet$ hold.

Action effects with inconsistent transition rewards are not permitted. For example, the effect `(probabilistic 0.5 (increase (reward) 1))` is semantically invalid because it associates a reward of both 1 and 0 to a self-transition.

Regular PDDL goals are used to express goal-type performance objectives. A goal statement `(:goal` ϕ`)` for a probabilistic planning problem encodes the objective that the probability of achieving ϕ should be maximized, unless an explicit optimization metric is specified for the planning problem.

For planning problems instantiated from a domain declaring the `:rewards` requirement, the default plan objective is to maximize the expected reward. A goal statement in the specification of a reward oriented planning problem identifies a set of absorbing states. In addition to transition rewards specified in action effects, it is possible to associate a one-time reward for entering a goal state. This is done using the `(:goal-reward` f`)` construct, where f is a numeric expression.

In general, a statement `(:metric maximize` f`)` in a problem definition means that the expected value of f should be maximized. PPDDL defines `goal-probability` as a special optimization metric that can be used to explicitly specify that the plan objective is to maximize (or minimize) the probability of goal achievement.

Formal Semantics

We present a formal semantics for PPDDL planning problems in terms of a mapping to a probabilistic transition system with rewards. A planning problem defines a set of state variables V, possibly containing both Boolean and numeric state variables. An assignment of values to state variables defines a state, and the state space S of the planning problem is the set of states representing all possible assignments of values to variables. In addition to V, a planning prob-

```
(define (domain coffee-delivery)
   (:requirements :negative-preconditions :disjunctive-preconditions
                  :conditional-effects :mdp)
   (:predicates (in-office) (raining) (has-umbrella) (is-wet)
                (has-coffee) (user-has-coffee))
   (:action buy-coffee
            :effect (and (when (not (in-office)) (probabilistic 0.8 (has-coffee)))
                         (when (user-has-coffee) (increase (reward) 0.8))
                         (when (not (is-wet)) (increase (reward) 0.2))))
   ...)
```

Figure 2: Part of PPDDL encoding of "Coffee Delivery" domain.

lem defines an initial-state distribution $p_0 : S \to [0, 1]$ with $\sum_{s \in S} p_0(s) = 1$ (i.e. p_0 is a probability distribution over states), a formula ϕ over V characterizing a set of goal states $G = \{s \mid s \models \phi\}$, a one-time reward r_G associated with entering a goal state, and a set of actions A instantiated from PPDDL action schemata. For goal-directed planning problems, without explicit rewards, we use $r_G = 1$.

An action $a \in A$ consists of a precondition ϕ_a and an effect e_a. Action a is applicable in a state s if and only if $s \models \phi_a$. It is an error to apply a to a state such that $s \not\models \phi_a$. This is consistent with the semantics of PDDL2.1 (Fox & Long 2003) and permits the modeling of forced chains of actions. Effects are recursively defined as follows (cf. Rintanen 2003):

1. $\top$ is the null-effect, represented in PPDDL by `(and)`.
2. b and $\neg b$ are effects if $b \in V$ is a Boolean state variable.
3. $x \leftarrow f$ is an effect if $x \in V$ is a numeric state variable and f is a real-valued function on numeric state variables.
4. $r \uparrow f$ is an effect if f is a real-valued function on numeric state variables.
5. $e_1 \wedge \ldots \wedge e_n$ is an effect if $e_1, \ldots, e_n$ are effects.
6. $c \rhd e$ is an effect if c is a formula over V and e is an effect.
7. $p_1 e_1 | \ldots | p_n e_n$ is an effect if $e_1, \ldots, e_n$ are effects, $p_i \geq 0$ for all $i \in \{1, \ldots, n\}$, and $\sum_{i=1}^{n} p_i = 1$.

Items 2 through 4 are referred to as *simple effect*. The effect b sets the Boolean state variable b to true in the next state, while $\neg b$ sets b to false in the next state. For $x \leftarrow f$, the value of f in the current state becomes the value of the numeric state variable x in the next state. Effects of the form $r \uparrow f$ are used to associate rewards with transitions as described below.

An action $a = \langle \phi_a, e_a \rangle$ defines a transition probability matrix P_a and a transition reward matrix R_a, with p^a_{ij} being the probability of transitioning to state j when applying a in state i, and r^a_{ij} being the reward associated with the state transition from i to j when caused by a. We can compute P_a and R_a by first translating e_a into an effect of the form $p_1 e_1 | \ldots | p_n e_n$, where each e_i is a deterministic effect. Rintanen (2003) calls this form Unary Nondeterminism Normal Form. Any effect e can be translated into this form by using the top four equivalences in Figure 3.

We further rewrite the effect of an action by translating each e_i into an effect of the form $(c_{i1} \rhd e_{i1}) \wedge \ldots \wedge (c_{in_i} \rhd e_{in_i})$, where each e_{ij} is a conjunction of simple effects and the conditions are mutually exclusive and exhaustive (i.e. $c_{ij} \wedge c_{ik} \equiv \bot$ for all $j \neq k$ and $\bigvee_{j=1}^{n_i} c_{ij} \equiv \top$). The bottom four equivalences in Figure 3 allow us to perform the desired translation.

An effect of the form $c \rhd e$, where e is a conjunction of simple effects, defines a set of state transitions. We assume that e is consistent. Actions with inconsistent effects are not valid PPDDL actions, and care should be taken when designing a PPDDL domain to ensure that no instantiations of action schemata can have inconsistent effects. A conjunction of simple effects is inconsistent if it contains both b and $\neg b$, or multiple *non-commutative* updates of a single numeric state variable. Two effects $x \leftarrow f$ and $x \leftarrow f'$ are commutative if $f(s[x = f'(s)]) = f'(s[x = f(s)])$, where $f(s)$ is the value of f evaluated in state s and $s[x = y]$ denotes a state with all state variables having the same value as in state s, except for x which has value y, i.e. numeric effects are commutative if they are insensitive to ordering. Under these assumptions, the following function can be defined:

$$\begin{aligned}
\tau(s, s', \top) &\doteq s' \\
\tau(s, s', b) &\doteq s'[b = \top] \\
\tau(s, s', \neg b) &\doteq s'[b = \bot] \\
\tau(s, s', x \leftarrow f) &\doteq s'[x = f(s)] \\
\tau(s, s', r \uparrow f) &\doteq s' \\
\tau(s, s', e_1 \wedge e_2) &\doteq \tau(s, \tau(s, s', e_1), e_2)
\end{aligned}$$

We can use τ to describe the set of state transitions defined by the effect $c \rhd e$:

$$T(c \rhd e) = \{\langle s, s' \rangle | s \models c \text{ and } s' = \tau(s, s, e)\}.$$

Given this definition of $T(c \rhd e)$, we can compute a transition matrix T_{ij} for each $c_{ij} \rhd e_{ij}$. The element at row s and column s' of T_{ij} is 1 if $\langle s, s' \rangle \in T(c_{ij} \rhd e_{ij})$, and 0 otherwise. Since we have ensured that the conditions c_{ij} are mutually exclusive, we get $P_a = \sum_{i=1}^{n} p_i T_i$ as the transition probability matrix for action a, where $T_i = \sum_{j=1}^{n_i} T_{ij}$. Finally, we need to make all states that satisfy the goal condition ϕ of the problem absorbing. This is accomplished by modifying P_a: for each s such that $s \models \phi$, we set the entry at row s and column s to 1 and the remaining entries on the same row to 0.

$$
\begin{aligned}
e \equiv & 1e \\
e \wedge (p_1 e_1 | \ldots | p_k e_n) \equiv & p_1(e \wedge e_1) | \ldots | p_n(e \wedge e_n) \\
c \rhd (p_1 e_1 | \ldots | p_n e_n) \equiv & p_1(c \rhd e_1) | \ldots | p_n(c \rhd e_n) \\
p_1(p'_1 e'_1 | \ldots | p'_k e'_k) | p_2 e_2 | \ldots | p_n e_n \equiv & (p_1 p'_1) e_1 | \ldots | (p_1 p'_k) e'_k | p_2 e_2 | \ldots | p_n e_n
\end{aligned}
$$

$$
\begin{aligned}
e \equiv & \top \rhd e \\
c \rhd e \equiv & (c \rhd e) \wedge (\neg c \rhd \top) \\
c \rhd (c' \rhd e) \equiv & (c \wedge c') \rhd e \\
(c_1 \rhd e_1) \wedge (c_2 \rhd e_2) \equiv & ((c_1 \wedge c_2) \rhd (e_1 \wedge e_2)) \wedge ((c_1 \wedge \neg c_2) \rhd e_1) \\
& \wedge ((\neg c_1 \wedge c_2) \rhd e_2) \wedge ((\neg c_1 \wedge \neg c_2) \rhd \top)
\end{aligned}
$$

Figure 3: Effect equivalences.

The reward associated with a conjunction of simple effects can be defined as follows:

$$
\begin{aligned}
r(s, \top) \doteq & 0 \\
r(s, b) \doteq & 0 \\
r(s, \neg b) \doteq & 0 \\
r(s, x \leftarrow f) \doteq & 0 \\
r(s, r \uparrow f) \doteq & f(s) \\
r(s, e_1 \wedge e_2) \doteq & r(s, e_1) + e(s, e_2)
\end{aligned}
$$

The effect $c_{ij} \rhd e_{ij}$ associates reward $r(s, e_{ij})$ with each transition $\langle s, s' \rangle \in T(c_{ij} \rhd e_{ij})$. We define a transition reward matrix R_{ij} for $c_{ij} \rhd e_{ij}$. The element at row s and column s' of R_{ij} is $r(s, e_{ij})$ for $s' = \tau(s, s, e_{ij})$ and 0 if $\langle s, s' \rangle \notin T_{ij}$. We then sum over all $c_{ij} \rhd e_{ij}$ to get a transition reward matrix for e_i: $R_i = \sum_{j=1}^{n_i} R_{ij}$.

The same transition may occur in multiple outcomes of the effect $p_1 e_1 | \ldots | p_n e_n$, and we require the reward for a specific transition to be consistent across outcomes. Let $\bullet$ represent the fact that the reward is undefined for a transition. We define $\tilde{R}_i$ to be R_i with an element at row s and column s' set to $\bullet$ if the element at row s and column s' of T_i is zero (i.e. e_i does not define a transition from s to s'). We define an element-wise matrix operator $\odot$ as follows:

$$
\begin{aligned}
\bullet \odot x \doteq & x \\
x \odot \bullet \doteq & x \\
x \odot x \doteq & x \\
x \odot y \doteq & \mathit{error} \text{ if } x \neq y
\end{aligned}
$$

We can now define the transition reward matrix for action a: $R_a = R_G + \bigodot_{i=1}^{n} \tilde{R}_i$. R_G represents the one-time reward associated with goal states. The entry at row s and column s' of R_G is set to r_G if $s \not\models \phi$ and $s' \models \phi$, and 0 otherwise. The transition reward matrix is well-defined if and only if the transition rewards are consistent across all outcomes of an action.

References

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89(1–2):219–283.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Ghallab, M.; Howe, A. E.; Knoblock, C. A.; McDermott, D.; Ram, A.; Veloso, M. M.; Weld, D. S.; and Wilkins, D. 1998. PDDL—the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. New York, NY: John Wiley & Sons.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1–2):239–286.

Littman, M. L. 1997. Probabilistic propositional planning: Representations and complexity. In *Proc. Fourteenth National Conference on Artificial Intelligence*, 748–754. Providence, RI: American Association for Artificial Intelligence.

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Rintanen, J. 2003. Expressive equivalence of formalism for planning with sensing. In Giunchiglia, E.; Muscettola, N.; and Nau, D. S., eds., *Proc. Thirteenth International Conference on Automated Planning and Scheduling*, 185–194. Trento, Italy: AAAI Press.