

Informatics 2D Tutorial 6

PDDL, State-Space Search and Partial-Order Planning

Matt Crosby and Alex Lascarides

Week 7

Task Description

You ask your personal robotic assistant to make you some tea. Your robot then needs to find a sensible plan that satisfies your request.

Part 1: Representing the World using PDDL

Use the Planning Domain Definition Language (PDDL) to represent your world.

1. You will first need to define the predicates that will then be used to describe the states in your world. You can define atemporal predicates that describe objects. Atemporal predicates do not change with the execution of actions. You can also define fluent predicates that can change with the execution of actions. To make tea you will need:
 - Water that can be either hot or cold (not hot), and can be in a bottle, kettle or a cup.
 - A kettle that can be empty or not empty.
 - A cup that can be empty or not empty.
 - A bottle that can be empty or not empty.
 - A tea bag that can either be in the cup or not in the cup.
2. Define an initial state where the tea bag is not in the cup, cold water is in the bottle, and all of the other containers are empty.
3. Formally describe the goal state of the plan. In this case the goal is to have tea (a cup full of hot water with a tea bag in it).
4. Define the following actions in terms of the *preconditions* that must hold prior to taking the action, and the *effects* that describe the changes made to the world after executing the action:

- Pour: An action that allows you to pour water from one container to another.
- AddTeaBag: An action that allows you to place a tea bag into the cup.
- BoilWaterInKettle: An action that allows you to use the kettle to boil the water.

Part 1: Solutions

1. For each object define an atemporal predicate:

$$water(x), bottle(x), kettle(x), cup(x), teabag(x)$$

For each property define a fluent predicate:

$$in(x, y), hot(x), empty(x)$$

Where $in(x, y)$ means x is inside y .

Atemporal predicates and fluent predicates have the same notation, however, atemporal predicates do not change as a result of actions while fluents do.

2. Initial State: $water(W) \wedge bottle(B) \wedge kettle(K) \wedge cup(C) \wedge teabag(T) \wedge in(W, B) \wedge empty(K) \wedge empty(C)$

Since we are using the closed world assumption we only describe what is true about the world. For example, we do not need to specify $\neg hot(W)$.

3. Goal State: $water(W) \wedge teabag(T) \wedge cup(C) \wedge in(W, C) \wedge hot(W) \wedge in(T, C)$.

The goal state could contain negations; however, in this case there aren't any.

4. The actions would be:

$Action(Pour(x, from, to))$

precondition: $water(x) \wedge in(x, from) \wedge \neg in(x, to) \wedge empty(to)$

effect: $\neg in(x, from) \wedge in(x, to) \wedge \neg empty(to) \wedge empty(from)$

In this case, we need to make sure that $from$ and to are either a cup, kettle or bottle, and want to avoid allowing the planner combine them with any other objects. Since writing this disjunctively is not allowed, we need to find a way around this. One way is to introduce a new predicate $vessel(x)$ that is initialised as true for objects of the type cup, kettle, and bottle, and then add a precondition that to and $from$ should be vessels, as follows:

$Action(Pour(x, from, to))$

precondition: $water(x) \wedge in(x, from) \wedge \neg in(x, to) \wedge empty(to) \wedge vessel(to) \wedge$

vessel(from)
 effect: $\neg in(x, from) \wedge in(x, to) \wedge \neg empty(to) \wedge empty(from)$
Action(AddTeaBag(t, c))
 precondition: $teabag(t) \wedge cup(c) \wedge \neg in(t, c)$
 effect: $in(t, c)$
Action(BoilWaterInKettle(w, k))
 precondition: $water(x) \wedge kettle(k) \wedge \neg hot(w) \wedge in(w, k)$
 effect: $hot(w)$

Part 2: Partial-Order Planning

Create a partial-order plan that takes you from the initial state to the goal state. Show which actions depend on one another and which actions do not. What is the advantage of partial-order planning over state-space search?

Part 2: Solutions

The plan branches out from the initial state *START* in two routes;

1) *START* \rightarrow *Pour(W, B, K)* \rightarrow *BoilWaterInKettle(W, K)* \rightarrow *Pour(W, K, C)* \rightarrow *FINISH*

2) *START* \rightarrow *AddTeaBag(T, C)* \rightarrow *FINISH*

Combining these two routes results in a complete plan from the *START* state to the goal. The arrows show the dependency between the actions.

The advantage of partial-order planning over state-space search has to do with the flexibility of the order of the actions. Another advantage is efficiency of the planning process as one partially ordered plan can represent several totally ordered ones, so you are effectively searching the plan space in parallel.