02285 AI and MAS, F25
Theory Assignment, Part 2
Due: 10 March at 20.00

Thomas Bolander, Mikkel Birkegaard Andersen

Important!

All conditions regarding collaboration, conventions and handing in is as for part 1.

In this exercise we consider how to represent the hospital domain in PDDL. At first we consider only the single-agent case, and ignore the colors, i.e., just assume all boxes to have the same color as the agent, so that we don't need to represent and keep track of the colors. Assume we are given a specific single-agent level. We will now show how to represent the configurations of this level as PDDL *states*, that is, conjunctions of literals. We choose *constants* to denote the entities in the level as follows:

- 0 denotes the agent.
- $B_1, B_2, \ldots, B_b$ is an enumeration of the boxes in the level.
- $G_1, G_2, \ldots, G_g$ is an enumeration of the goals in the level.
- $L_1, L_2, \ldots, L_l$ is an enumeration of the locations (grid cells) in the level, excluding walls.

Then a *state* of the level is a conjunction of the form:

$$\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{G} \wedge \mathcal{F} \wedge \mathcal{L}_1 \wedge \mathcal{L}_2 \wedge \mathcal{N},$$

where:

$$\begin{aligned}
\mathcal{A} &= \mathit{AgentAt}(0, L_i), \text{ where } L_i \text{ is the current location of agent 0} \\
\mathcal{B} &= \bigwedge\{\mathit{BoxAt}(B_i, L_j) \mid \text{box } B_i \text{ is at location } L_j\} \\
\mathcal{G} &= \bigwedge\{\mathit{GoalAt}(G_i, L_j) \mid \text{goal } G_i \text{ is at location } L_j\} \\
\mathcal{F} &= \bigwedge\{\mathit{Free}(L_i) \mid \text{location } L_i \text{ is a free cell}\} \\
\mathcal{L}_1 &= \bigwedge\{\mathit{Type}(G_i, \alpha) \mid \alpha \in \{A, B, \ldots, Z\} \text{ is the type of goal } G_i\} \\
\mathcal{L}_2 &= \bigwedge\{\mathit{Type}(B_i, \alpha) \mid \alpha \in \{A, B, \ldots, Z\} \text{ is the type of box } B_i\} \\
\mathcal{N} &= \bigwedge\{\mathit{Neighbour}(L_i, L_j) \mid L_i \text{ and } L_j \text{ are neighbours}\}
\end{aligned}$$

The *initial state* is the one in which $\mathcal{A}$, $\mathcal{B}$, $\mathcal{G}$ and $\mathcal{F}$ are defined by the initial locations of the agent, boxes, goals, and free cells. The corresponding *goal* is:

$$\bigwedge_{i=1,\ldots,g} \mathit{GoalAt}(G_i, x_i) \wedge \mathit{BoxAt}(y_i, x_i) \wedge \mathit{Type}(y_i, z_i) \wedge \mathit{Type}(G_i, z_i),$$

where all x_i, y_i and z_i are variables. Note that by definition of PDDL, a goal formula is satisfied if its *existential closure* is satisfied, that is, if there exists constants to instantiate the variables x_i, y_i and z_i such that the resulting formula is true in the state. Note also that role of the conjunctions $\mathit{Type}(y_i, z_i) \wedge \mathit{Type}(G_i, z_i)$ is to make sure that boxes end up at goal cells of the correct type.

ACTION : $Move(agt, agtfrom, agtto)$

PRECONDITION : AgentAt(agt, agtfrom) ∧ Neighbour(agtfrom, agtto) ∧ Free(agtto) ∧
∧ ∧ ∧
∧ ∧

EFFECT : -AgentAt(agt, agtfrom) ∧ AgentAt(agt, agtto) ∧ Free(agtfrom) ∧
-Free(agtto) ∧ ∧ ∧
∧ ∧ ∧

ACTION : $Push(agt, agtfrom, box, boxfrom, boxto)$

PRECONDITION : AgentAt(agt, agtfrom) ∧ BoxAt(box, boxfrom) ∧ Neighbour(agtfrom, boxfrom) ∧
Neighbour(boxfrom, boxto) ∧ Free(boxto) ∧ ∧
∧ ∧ ∧

EFFECT : -AgentAt(agt, agtfrom) ∧ AgentAt(agt, boxfrom) ∧ -BoxAt(box, boxfrom) ∧
BoxAt(box, boxto) ∧ Free(agtfrom) ∧ -Free(boxto) ∧
∧ ∧ ∧

ACTION : $Pull(agt, agtfrom, agtto, box, boxfrom)$

PRECONDITION : AgentAt(agt, agtfrom) ∧ BoxAt(box, boxfrom) ∧ Neighbour(agtfrom, boxfrom) ∧
Neighbour(agtfrom, agtto) ∧ Free(agtto) ∧ ∧
∧ ∧ ∧

EFFECT : -AgentAt(agt, agtfrom) ∧ AgentAt(agt, agtto) ∧ -BoxAt(box, boxfrom) ∧
BoxAt(box, agtfrom) ∧ Free(boxfrom) ∧ -Free(agtfrom) ∧
∧ ∧ ∧

Figure 1: Action schemas for the single-agent hospital domain

Figure 2: A very simple level (left) and its corresponding location enumeration (right).

Q1 Action schemas for the hospital domain

Fill in the preconditions and effects of the action schemas in Figure 1 so that they formalise the actions available in single-agent domains. We use the following naming conventions:

- *agt* is the agent (we are so far only considering single-agent levels, so this parameter can only be instantiated with 0).
- *agtfrom* is the location of the agent before the action is executed.
- *agtto* is the location of the agent after the action is executed.
- *box* is the box to be moved (in the *Push* and *Pull* actions).
- *boxfrom* is the location of the box before the action is executed.
- *boxto* is the location of the box after the action is executed.

Note that we are not using the directions (W, E, S and N) as parameters in these action schemas. They are not needed when we have the exact locations as parameters. The action schemas you design for these three actions should of course match exactly how these are specified in the hospital domain, cf. the description in `hospital_domain.pdf`. Make sure to use a *minimal number of literals*, that is, never add a precondition or effect literal that is not necessary to include.

Note that the conjunction symbols are already included in the figure, so you should just put one literal in each of the blank text fields. You might not need all text fields, and in this case you should just use the first ones, and leave the remaining ones blank. Recall that you should use `-` for the negation sign. Make sure to only use the predicate and variable names introduced above, and make sure to spell them exactly as stated. Thus `Neighbour(agtfrom,agtto)` is correct, but neither `Neighbor(agtfrom,agtto)` nor `Neighbour(agtfrom,agto)` is.

Q2 Computing plans in the hospital domain

Consider the very simple level of Figure 2. A plan to solve the level is the following, using the names $L1, \ldots, L4$ for the locations as provided in Figure 2 and $B1$ for the box:

$$\mathit{Move}(0, L3, L2), \mathit{Pull}(0, L2, L3, B1, L4), \mathit{Push}(0, L3, B1, L2, L1).$$

Compute the sequence of states that the agent will pass through when executing this plan, using your action schemas from the previous question. Some of the literals are rigid, i.e., will never change as the consequence of performing an action. These are the literals in $\mathcal{G}$, $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{N}$. We don't need to keep track of those, so when you compute the sequence of states, you only need to specify the true atoms in $\mathcal{A} \cup \mathcal{B} \cup \mathcal{F}$. Fill in Figure 3 with your computed sequence of states. As before, leave unused text fields blank. Note also again that the conjunction symbols have already been provided, so only enter one literal in each text field.

Before moving on, check that the state reached after having executed the entire plan satisfies the goal, using the goal formula given in the beginning of the exercise.

Initial state:

AgentAt(0, L3) ∧ BoxAt(B1, L4) ∧ Free(L1) ∧ Free(L2)

After *Move* action:

AgentAt(0, L2) ∧ BoxAt(B1, L4) ∧ Free(L1) ∧ Free(L3)

After *Pull* action:

AgentAt(0, L3) ∧ BoxAt(B1, L2) ∧ Free(L1) ∧ Free(L4)

After *Push* action:

AgentAt(0, L2) ∧ BoxAt(B1, L1) ∧ Free(L3) ∧ Free(L4)

Figure 3: State sequence of plan.

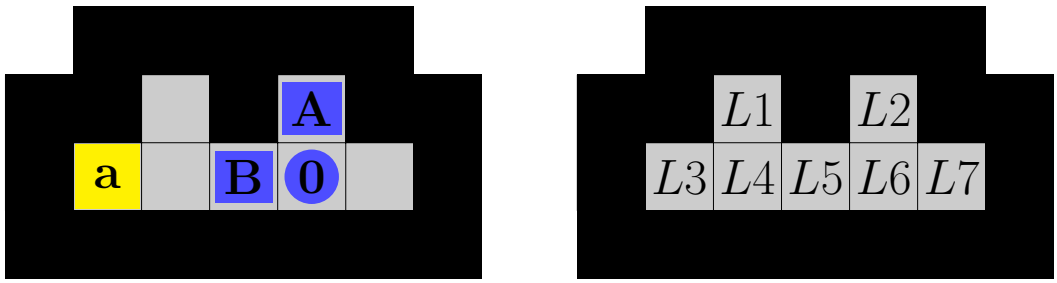

Figure 4: A single-agent hospital level (left) and a corresponding location enumeration (right). We can use $B1$ to denote the A box, and $B2$ to denote the B box.

The goal formula has $g = 1$ in this case, since there is only one goal cell. This means the goal formula contains three variables, x_1, y_1 and z_1. As earlier mentioned, a goal state is one in which the existential closure of the goal formula is satisfied, i.e., there has to be a way to instantiate all the variables so that the resulting ground formula is true in the state. Enter below the constants that the variables have to be instantiated with for the goal formula to be true in the state reached by the plan above.

1. The variable x_1 of the goal formula has to be instantiated with the constant L1
2. The variable y_1 of the goal formula has to be instantiated with the constant B1
3. The variable z_1 of the goal formula has to be instantiated with the constant A

Q3 IW algorithm

Suppose that when expanding a state, we always add the child nodes to the frontier in lexicographic order according to the action that generated the child. This means that for instance the result of *Move* actions will always be added to the frontier before the result of *Pull* actions, and the result of a $Pull(0, L2, L1, B1, L4)$ will be added before the result of a $Pull(0, L2, L3, B1, L4)$ (because $L1$ is lexicographically smaller than $L2$). Perform $IW(1)$ on the planning problem of Figure 2 by drawing the full search tree that the algorithm will build. Make sure to highlight the new atoms in each generated child state, so it becomes clear why that child state is not blocked/pruned (cf. the example by Frederik Drachmann on the slides).

1. The number of states in the search tree built by $IW(1)$ is (where the search tree consists of the initial state as well as any state that is added to the frontier during the run of the algorithm) 6

2. The length of the solution found by $IW(1)$ is (write `infinity` if no solution was found) 3

3. No matter in which order we add child states to the frontier, $IW(1)$ will necessarily find the optimal solution. ☑ True ☐ False

4. No matter in which order we add child states to the frontier, $IW(1)$ will necessarily find *some* solution. ☐ True ☑ False

5. For some order of adding child states to the frontier, $IW(1)$ will find an optimal solution. ☑ True ☐ False

6. The problem has width at most 2. ☐ True ☑ False

7. The problem has width 2. ☑ True ☐ False

8. For some order of adding child states to the frontier, the effective width of the problem is 2. ☐ True ☑ False

9. No matter in which order we add child states to the frontier, $IW(2)$ will necessarily find the optimal solution. ☑ True ☐ False

10. Any solvable single-agent planning problem in the hospital domain with a single agent and a single box has width at most 2. ☐ True ☑ False

11. Any solvable single-agent planning problem in the hospital domain with a single agent and a single box will be solved optimally by $IW(2)$, independently of the order in which we add child states to the frontier. ☐ True ☑ False

12. Consider the level shown in Figure 4. It has width 2. ☑ True ☐ False

Q4 Extending with colors

Now consider extending the problem definition to deal with levels containing multiple agents. At first, consider only extending the domain to deal with actions composed asynchronously, that is, one agent acting at a time. $\mathcal{A}$ has to specify initial location of all agents. We also have to add colors to the initial state:

$$\mathcal{C} = \bigwedge\{Color(\alpha, \beta) \mid \alpha \text{ is agent or box and } \beta \text{ is its color}\}.$$

Define the revised action schemas for *Move*, *Push* and *Pull* in the setting of multiple agents with only one agent acting at a time. Then answer the following questions about the revised action schemas.

1. The original action schema for *Move* used 3 variables: *agt*, *agtfrom* and *agtto*. How many additional variables does the revised action schema use? 0

2. How many additional precondition literals does the revised *Move* action have? 0

3. How many additional effect literals does the revised *Move* action have? 0

4. The original action schema for *Push* used 5 variables. How many additional variables does the revised action schema use? 1

5. How many additional precondition literals does the revised *Push* action have? 2

6. How many additional effect literals does the revised *Push* action have? 0

7. The original action schema for *Pull* used 5 variables. How many additional variables does the revised action schema use? 1

8. How many additional precondition literals does the revised *Pull* action have? 2

9. How many additional effect literals does the revised *Pull* action have? 0

Q5 Extending to synchronous actions

Now consider extending the problem definition to deal with multiple agents acting concurrently (MA track). A *joint action* is of the form

$$a_0 \mid \cdots \mid a_n$$

where a_0 is the action of agent 0, a_1 is the action of agent 1, etc. This implies that a_0 is an action having 0 in its first parameter, that is, of the form $Move(0,\dots)$, $Push(0,\dots)$ or $Pull(0,\dots)$. Similarly, a_1 is an action having 1 in its first parameter, etc. Note that this is different from the programming project in which the agent parameter has been made implicit when joint actions are sent to the server.

We define the following conflict types between pairs of actions a_i, a_j:

CellConflict(a_i, a_j) Actions a_i and a_j attempt to move two distinct objects (agents or boxes) into the same cell.

BoxConflict(a_i, a_j) Actions a_i and a_j attempt to move the same box.

By definition of the hospital domain (see `hospital_domain.pdf`), two actions a_i of agent i and a_j of agent $j \neq i$ are *conflicting* if either *CellConflict*(a_i, a_j) or *BoxConflict*(a_i, a_j) holds.

The goal of this question is to investigate whether it is possible to reduce the problem of checking for conflict between two actions a_i and a_j to a condition expressed exclusively in terms of the logical properties of the corresponding action schemas. Recalling the standard notions from logic, a conjunction of literals A is *inconsistent* with a conjunction of literals B if some literal occurs positively in one of the conjunctions and negatively in the other. In that case we also say that the two conjunctions are *mutually inconsistent*. For each of the statements below concerning the relation between conflicts and the action schemas of the individual actions, determine whether it is true or false. Mark the true statements. Everywhere it is assumed that a_i is an action of agent i, a_j is an action of agent $j \neq i$, and both a_i and a_j are (independently) applicable actions.

1. If *CellConflict*(a_i, a_j) holds then the effects of a_i and a_j are mutually inconsistent.
   ☐ True ☑ False

2. If *CellConflict*(a_i, a_j) holds then the precondition of one of the actions is inconsistent with the effect of the other. ☐ True ☑ False

3. If *BoxConflict*(a_i, a_j) holds then the effects a_i and a_j are mutually inconsistent.
   ☐ True ☑ False

4. If $BoxConflict(a_i, a_j)$ then the precondition of one of the actions is inconsistent with the effect of the other. ☐ True ☑ False

5. For all applicable actions a_i and a_j where the precondition of one is inconsistent with the effect of the other, either $CellConflict(a_i, a_j)$ or $BoxConflict(a_i, a_j)$ holds. ☐ True ☑ False

6. It is possible to reduce the problem of checking for conflict between two applicable actions a_i and a_j to a condition expressed exclusively in terms of consistency constraints on the action schemas (whether precondition/effects are mutually inconsistent or not).

   ☐ True ☑ False