02285 AI and MAS, SP18
Theory Assignment
Due: Monday 5 March at 20.00

Thomas Bolander

This assignment is to be carried out in **groups of 2–3 students**, with 2 as the recommended number. Your solution is to be handed in via CampusNet: Go to Assignments, then choose Theory Assignment. You should only hand in a single pdf file with your solutions to the exercises. Remember to put your full names, student numbers, course number, assignment name ("Theory Assignment") on the front page of your submission.

Exercise 1 (Painting domain: action schemas, state spaces, POP algorithm, domain independent heuristics)

Consider the following painting planning domain. You're given a number of cans of paint, each of which contain paint of a certain colour. There is a brush, which can be dipped in a can, after which it can be used to paint a block. The available actions are:

ACTION : $DipBrush(r, c, k)$
PRECONDITION : $Brush(r) \land Can(c) \land IsColor(k) \land Color(c, k)$
EFFECT : $CanPaint(r, k)$

ACTION : $Paint(b, r, k)$
PRECONDITION : $Block(b) \land Brush(r) \land IsColor(k) \land CanPaint(r, k)$
EFFECT : $Color(b, k) \land \neg CanPaint(r, k)$

1. The action schema above can be simplified. We can remove the $IsColor(k)$ atom in both preconditions as well as the $Brush(r)$ precondition in $Paint(b, r, k)$, hence getting:

   ACTION : $DipBrush(r, c, k)$
   PRECONDITION : $Brush(r) \land Can(c) \land Color(c, k)$
   EFFECT : $CanPaint(r, k)$

   ACTION : $Paint(b, r, k)$
   PRECONDITION : $Block(b) \land CanPaint(r, k)$
   EFFECT : $Color(b, k) \land \neg CanPaint(r, k)$

Explain why it is possible to make these simplifications (you can make any necessary assumptions about the construction of the initial state). Also explain why the $Can(c)$ precondition in $DipBrush(r, c, k)$ can *not* be omitted.

2. What are the rigid atoms in this planning domain? (cf. the weekly exercises).

3. Consider a planning problem in the painting domain with initial state given by

$$s_0 = Block(B_1) \wedge Block(B_2) \wedge Can(C_1) \wedge Brush(R_1) \wedge Color(C_1, Green).$$

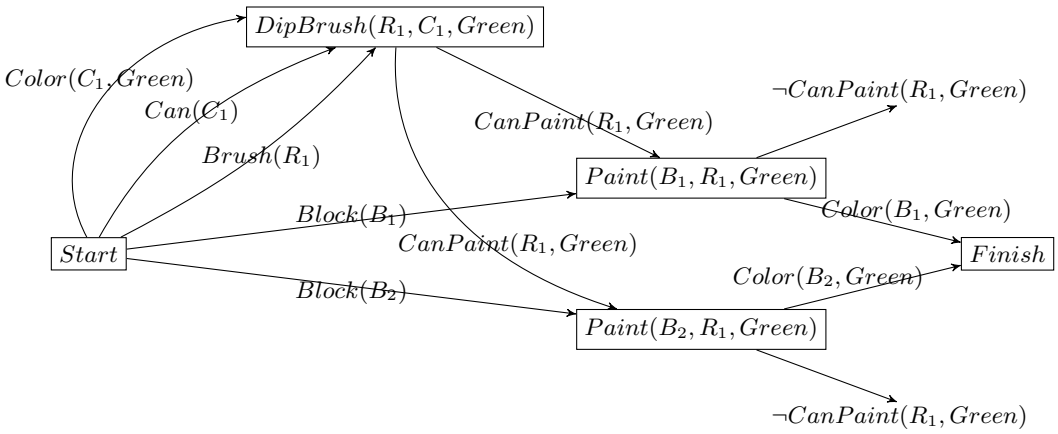
Draw the entire state space reachable from this initial state. Omit the rigids when drawing states. Remember to label all edges by their corresponding actions.

4. Use the state space above to find an optimal plan for the planning problem with initial state s_0 (given above) and goal given by

$$g = Color(B_1, Green) \wedge Color(B_2, Green).$$

You can either complete this question by marking an optimal plan as a path on the state space above (using a different color), or by listing the sequence of actions in such a plan.

5. Consider the following attempt to create a partially ordered plan for the planning problem of the previous question:



Briefly explain why this is *not* a solution according to the POP algorithm, that is, why it doesn't constitute a partially ordered plan with no open preconditions and no unresolved conflicts.

6. Use the POP algorithm to find a correct solution to the same planning problem as above (the goal is $Color(B_1, Green) \wedge Color(B_2, Green)$). You should illustrate and explain your steps and how you resolve conflicts and/or backtrack, not only present the final graph!

7. How many linearisations does your partial-order plan have? How many distinct solutions does the problem itself have? How many optimal solutions does it have? Remember to briefly explain how you found those numbers. For the last two questions, you can e.g. refer to the state space constructed earlier.

8. Argue that $h_{gc}(s_0) = h_{ip}(s_0) = 2$ for the same planning problem as above.

9. Determine $h^+(s_0)$ for the same planning problem as above, that is, having

$$s_0 = Block(B_1) \wedge Block(B_2) \wedge Can(C_1) \wedge Brush(R_1) \wedge Color(C_1, Green)$$
$$g = Color(B_1, Green) \wedge Color(B_2, Green)$$

   h^+ is calculated from the relaxed painting problem where negative action effects are ignored. Explain in words what relaxation of the painting problem this corresponds to.

10. Determine $h_{add}(s_0)$ and $h_{max}(s_0)$ for the same planning problem as above. Show your calculations.

11. Consider again the planning problem from above. First show that $h_{add}(s_0) = h^*(s_0)$. Then find another initial state s_0' satisfying $h_{add}(s_0') < h^*(s_0')$ (with the same goal g), that is, find an initial state for which the additive heuristics is (too) optimistic.

12. Consider the following action sequence:

$$DipBrush(R_1, C_1, Yellow), Paint(B_1, R_1, Yellow),$$
$$DipBrush(R_1, C_2, Blue), Paint(B_2, R_1, Blue).$$

    The result will be that the block B_1 becomes yellow, and block B_2 blue. However, in real life, B_2 might become slightly greenish, as a bit of the yellow paint is bound to be still on the brush when it is dipped into the blue can. Usually one would clean a brush whenever the brush has to be used for a new color. We will now describe action schemas for capturing this.

    We can modify the action schemas so that a brush can not be dipped in another color before it has been cleaned (using a $CleanBrush$ action). We then have to split $CanPaint(r, k)$ into two atoms: $Color(r, k)$ and $CanPaint(r)$. $Color(r, k)$ is used to record the latest color k the brush r has been dipped in. $CanPaint$ is used to record whether the brush is wet, that is, it has been dipped but not yet used to paint anything. We also add a new predicate $Clean(r)$ for whether r is clean.

Then the resulting action schemas become:

ACTION : $DipCleanBrush(r, c, k)$
PRECONDITION : $Brush(r) \wedge Can(c) \wedge Color(c, k) \wedge Clean(r)$
EFFECT : $CanPaint(r) \wedge Color(r, k) \wedge \neg Clean(r)$

ACTION : $DipBrush(r, c, k)$
PRECONDITION : $Brush(r) \wedge Can(c) \wedge Color(c, k) \wedge Color(r, k)$
EFFECT : $CanPaint(r)$

ACTION : $Paint(b, r, k)$
PRECONDITION : $Block(b) \wedge CanPaint(r) \wedge Color(r, k)$
EFFECT : $Color(b, k) \wedge \neg CanPaint(r, k)$

ACTION : $CleanBrush(r, k)$
PRECONDITION : $Brush(r) \wedge Color(r, k)$
EFFECT : $Clean(r) \wedge \neg Color(r, k)$

Briefly explain the following:

- Why are both $Color(c, k)$ and $Color(r, k)$ preconditions of $DipBrush$?
- Why do we need two distinct action schemas for dipping a brush, $DipCleanBrush$ and $DipBrush$?
- What would happen if we removed the negative effect $\neg Color(r, k)$ from $CleanBrush(r, k)$.

13. Provide an optimal plan using the action schemas from the previous question in the planning problem having:

- Two brushes R_1 and R_2, both initially clean;
- A can C_1 with $Green$ paint, a can C_2 with Red paint and a can C_3 with $Blue$ paint;
- Four blocks $B_1, \ldots, B_4$, initially black;
- The goal is for blocks B_1 and B_2 to be green, block B_3 to be red and block B_4 to be blue.

14. Consider again the planning domain defined by the modified action schemas from question 12. Assume the goal is

$$g = Color(B_1, Green) \wedge Color(B_2, Green).$$

Find an initial state s'_0 such that $h_{add}(s'_0) > h^*(s'_0)$, that is, an initial state for which the additive heuristics is (too) pessimistic. This should be contrasted with the result in question 11. Explain how it can be that the subgoal independence assumption in this domain in some cases lead to optimistic estimates ($h_{add}(s'_0) < h^*(s'_0)$) and in others to pessimistic estimates ($h_{add}(s'_0) > h^*(s'_0)$)?

Exercise 2 (Action schemes for the programming project)

In this exercise we consider how to express the problem given in the programming project (`prog_proj_assignment.pdf`) as a PDDL planning problem.

At first we consider only the single-agent case (SA track). Assume we are given a specific level in the SA track. 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:

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

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:

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

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 $Letter(y_i, z_i) \wedge Letter(G_i, z_i)$ is to make sure that boxes end up at goal cells with the correct letter.

1. Write the *action schemas* corresponding to the actions available in the domain. The actions should have the following names and parameters:

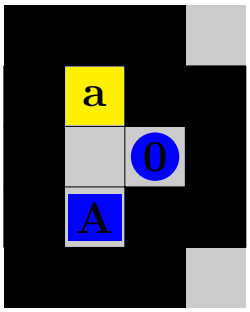
   $Move(agt, agtfrom, agtto)$
   $Push(agt, agtfrom, box, boxfrom, boxto)$
   $Pull(agt, agtfrom, agtto, box, boxfrom)$

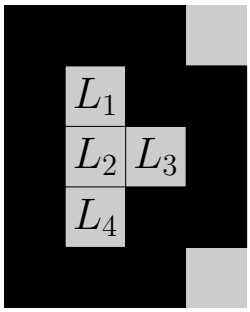
where

- 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 actions work in the programming project, cf. the description in `prog_proj_assignment.pdf`.

2. Consider the following very simple level (left) and its corresponding location enumeration (right):





A plan to solve the level is the following, using the action names above:

$$Move(0, L_3, L_2), Pull(0, L_2, L_3, B_1, L_4), Push(0, L_3, B_1, L_2, L_1).$$

Show the sequence of states (conjunctions of literals) that the agent will pass through when executing this plan, using your action schemas from the previous question. Some of the literals are rigid, that is, will never change as the consequence of performing an action. These are the literals in $\mathcal{G}$, $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{N}$. You only have to specify these literals in the initial state, not the following states. Show that the state reached after having executed the entire plan satisfies the goal, using the goal formula given in the beginning of the exercise.

3. 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. Describe the modifications required on the *initial state* and *action schemas*. *Hint*: To describe multi-agent levels you need to introduce colors.

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

$$[a_0, \ldots, a_n]$$

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 $Move(0, \ldots)$, $Push(0, \ldots)$ or $Pull(0, \ldots)$. 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 send to the server.

A joint action $[a_0, \ldots, a_n]$ is said to be *applicable* in a state s if it can executed successfully in that state, that is, if the following holds:

- Each of the individual actions $a_0, \ldots, a_n$ has its precondition satisfied in the state s.
- No pair of actions a_i and a_j are *conflicting*.

Let a_i be an action of agent i and a_j an action of agent j. The two actions are *conflicting* if:

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

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

Prove that two actions are conflicting if and only if the precondition of one of the actions is inconsistent with the effect of the other (using the action schemas you found above).

5. Consider adding a new agent type, *magicians*, to the domain. If agent i is a magician, $Magician(i)$ is added as a conjunct to the initial state. Magicians can change the color of any box in a neighbouring cell. Provide the action schema for an action $ChangeColor$ allowing a magician to change the color of a neighbouring box. You can assume that the initial state also contains atoms of the form $Color(\alpha)$ for each color α occurring in the level.

6. Provide an example of a MA track level (using the textual level format described in `prog_proj_assignment.pdf`) where only agent 0 is a magician and where:

(a) The level is solvable without using $ChangeColor$.

(b) The optimal solution using the magician to change colors is significantly shorter than the optimal solution without.

Note that if additional features like magicians were added to the domain, domain-specific heuristics and HTN approaches for the original domain would probably have to be revised quite significantly. If we decided to use many domain variations like magicians, teleporters etc. in the programming project, it would probably be best to go for an approach that uses a generic planner with domain-independent heuristics. However, when the domain is fixed and no variations are considered, domain-specific heuristics and approaches will generally be able to beat the generic ones.