02285 AI and MAS, F25
Theory Assignment, Part 1
Due: 3 March at 20.00

Thomas Bolander, Mikkel Birkegaard Andersen

Important!

Collaboration policy. This assignment is **individual** and should be completed by you and you alone. You are not allowed to share any part of your solutions with anyone, and not allowed to see any part of the solutions of anyone else. Violations of these conditions will be considered as violations of the academic honesty and will be reported to DTU.

Write notes. When doing the exercises, describe all your calculations, graphs, reasoning and intermediate results in a separate file (the one you hand in as `notes.pdf`, see below). No particular format is required. I suggest using pen and paper for this, and then scan the notes afterwards, as pen and paper makes it easier and faster to make drawings etc. However, you are allowed to use electronic tools for your notes, if you prefer. Many questions refer back to previous ones, so you will need to be able to go back and look at the details of those. Furthermore, many questions require a good deal of calculating and drawing, and will be much harder to complete without well-structured notes on the side.

Filling in the pdf. Don't add any text or comments to this pdf, only the answers to the questions in the **assigned boxes** (check boxes and fillable text fields). The pdf file is a fillable form, so use a pdf tool like e.g. Adobe Acrobat that supports fillable forms.

> **We will extract, process and assess the answers automatically using a script, so it's extremely important to follow all conventions (see below). Any spelling mistake will hence also be counted as an error, and it's your responsibility to make sure to follow all syntactic conventions.**

Save the file by using the normal save function in your pdf reader, **not** by printing to a file.

Conventions for filling in the text fields Some of the questions require you to enter a number. In the case that number is ∞, just write `infinity` in letters. Other questions require you to fill in names of literals or actions. Everything will be turned into lowercase characters with whitespace ignored, so e.g. `ColorOfBlock(B1, R)` and `colorofblock(b1,r)` are the same. The negation symbol is the `-` symbol, for instance `-ColorOfBlock(B1,G)`.

Handing in. This assignment is to be handed in as the "Theory Assignment, Part 1" on DTU Learn. You are required to hand in **two** pdf files:

1. A filled-in version of this pdf with the file name unchanged.
2. You `notes.pdf` file, see above. The notes will not be taken into account in assessing the Theory Assignment, except if you are between two grades for the final assessment of the course. In that case, the notes might be used to see whether the assessment of the Theory Assignment could be a bit higher, e.g. if you show the correct understanding despite a wrong result.

Consider the following painting planning domain. You're given a number of cans of paint and a number of brushes that can be dipped in the cans, after which they can be used to paint blocks. A planning problem in this domain will contain an initial state describing a number of blocks, $B_1, \ldots, B_b$, a number of brushes, $R_1, \ldots, R_r$, a number of cans, $C_1, \ldots, C_c$ and a number of colors $K_1, \ldots, K_k$. Each can contains paint of a specific color, also described in the initial state. Hence the initial state will contain the following literals:

- $\mathit{Block}(B_i)$ for each block B_i
- $\mathit{Brush}(R_i)$ for each brush R_i
- $\mathit{Can}(C_i)$ for each can C_i
- $\mathit{IsColor}(K_i)$ for each color K_i
- $\mathit{ColorInCan}(C_i, K_j)$ when the color of the paint in can C_i is K_j

A goal in this domain will be any conjunction of $\mathit{ColorOfBlock}(b, k)$ literals. The available actions are:

ACTION : $\mathit{DipBrush}(r, c, k)$
PRECONDITION : $\mathit{Brush}(r) \wedge \mathit{Can}(c) \wedge \mathit{IsColor}(k) \wedge \mathit{ColorInCan}(c, k)$
EFFECT : $\mathit{CanPaint}(r, k)$

ACTION : $\mathit{Paint}(b, r, k)$
PRECONDITION : $\mathit{Block}(b) \wedge \mathit{Brush}(r) \wedge \mathit{IsColor}(k) \wedge \mathit{CanPaint}(r, k)$
EFFECT : $\mathit{ColorOfBlock}(b, k) \wedge \neg \mathit{CanPaint}(r, k)$

Note that, by assumption, none of the $\mathit{ColorOfBlock}(b, k)$ atoms are true in the initial state, hence all blocks are initially uncolored. Also, none of the $\mathit{CanPaint}(r, k)$ atoms are true in the initial state, so initially none of the brushes can paint in any color.

Q1 Rigids

What are the rigid predicates of the planning domain? (cf. the weekly exercises). Mark the rigid predicates below:

- [x] Block
- [x] Brush
- [x] Can
- [ ] $\mathit{CanPaint}$
- [x] $\mathit{ColorInCan}$
- [ ] $\mathit{ColorOfBlock}$
- [x] $\mathit{IsColor}$

Q2 Simplified action schemas

The preconditions of the action schemas above can be simplified. We can remove some of the precondition literals without affecting the set of solutions that each planning problem in the domain has. Determine all the precondition literals that can be removed without affecting the set of solutions of each planning problem in the domain. Then mark below the precondition literals that still need to be **included** after the simplification.

ACTION : $DipBrush(r, c, k)$

PRECONDITION : ☑ $Brush(r) \wedge$ ☐ $Can(c) \wedge$ ☐ $IsColor(k) \wedge$ ☑ $ColorInCan(c, k)$

EFFECT : $CanPaint(r, k)$

ACTION : $Paint(b, r, k)$

PRECONDITION : ☑ $Block(b) \wedge$ ☐ $Brush(r) \wedge$ ☐ $IsColor(k) \wedge$ ☑ $CanPaint(r, k)$

EFFECT : $ColorOfBlock(b, k) \wedge \neg CanPaint(r, k)$

In the rest of this exercise, assume that the original action schemas have been replaced by the simplified ones.

Q3 State space

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

$$s_0 = Block(B1) \wedge Block(B2) \wedge Brush(R1) \wedge Can(C1) \wedge IsColor(G) \wedge ColorInCan(C1, G)$$

Here G is short for '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. Then answer the following questions concerning the state space:

1. The total number of states is 8
2. The total number of loop edges (edges with the same start and end point) is 4
3. The total number of edges, including loop edges, is 16
4. The minimal number of outgoing edges from any state is 1
5. The maximal number of outgoing edges from any state (branching factor) is 3

Q4 Solutions to planning problems

Let $\mathcal{A}$ denote your simplified set of action schemas, let s_0 be the initial state given above, and let

$$g = ColorOfBlock(B1, G) \wedge ColorOfBlock(B2, G).$$

Answer the following, using your state space of the previous question:

1. The length of an optimal solution to the planning problem $(\mathcal{A}, s_0, g)$ is 4
2. The number of optimal solutions to the planning problem $(\mathcal{A}, s_0, g)$ is 2
3. For any goal formula g', the maximal number of optimal plans to the planning problem $(\mathcal{A}, s_0, g')$ is 2

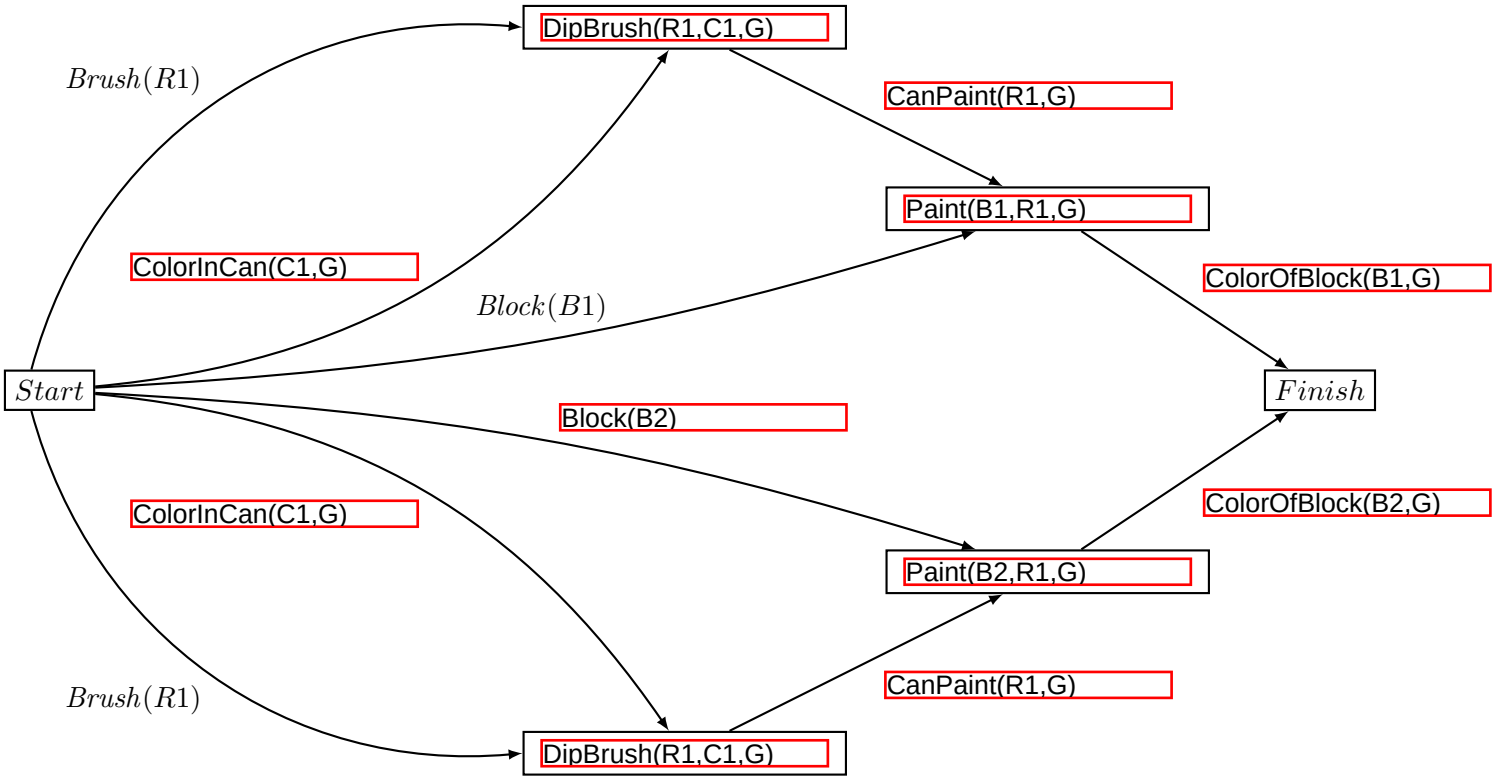

Figure 1: A partially ordered plan (POP graph).

Q5 POP algorithm

Consider the same planning problem $(\mathcal{A}, s_0, g)$ as above. The graph of Figure 1 is the skeleton of a partially ordered plan resulting from a run of the POP algorithm on this planning problem. The graph includes all the labelled ordering constraints (causal links), however not the unlabelled ones (the ones resulting from conflict resolution). Fill in both the names of the states and the labels on the labelled ordering constraints below. Three of the labels have already been filled in.

Now answer the questions below, where we define conflict as follows: An action $D \neq B$ is said to **conflict** with an ordering constraint $A \overset{c}{\prec} B$ if D's effect is inconsistent with c. A conflict is **resolved** by adding either the constraint $B \prec D$ or $D \prec A$, while making sure that the graph is still acyclic (so that $\prec$ is always a strict partial order).

1. How many conflicts are there in total in the partially ordered plan that you filled in? 2

2. How many distinct ways are there to resolve all conflicts in the partially ordered plan, that is, how many distinct conflict-free partially ordered plans can be constructed by conflict resolution? 2

3. Suppose you resolve all conflicts in the partially ordered plan. How many linearisations does your resulting partially ordered plan then have? 2

Q6 Domain-independent heuristics

We are now going to consider some of the different domain-independent heuristics for the planning problem $(\mathcal{A}, s_0, g)$ considered in Q4. Determine the following heuristic values:

1. $h_{gc}(s_0) =$ 2
2. $h_{ip}(s_0) =$ 2
3. $h^+(s_0) =$ 3
4. $h_{add}(s_0) =$ 4
5. $h_{max}(s_0) =$ 2
6. $h_{FF}(s_0) =$ 3

Q7 Modifying the domain

Consider the following action sequence:

$$DipBrush(R1, C1, Y), Paint(B1, R1, Y),$$
$$DipBrush(R1, C2, B), Paint(B2, R1, B).$$

Here Y is short for 'yellow' and B is short for 'blue' (not to be confused with $B1$ and $B2$). The result will be that the block $B1$ becomes yellow, and block $B2$ blue. However, in real life, $B2$ 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. The goal of this question is to describe action schemas for capturing this.

We call a brush *unclean* when it hasn't been cleaned since last dipped in a can. We can modify the action schemas so that an unclean brush can not be dipped in another color until it has been cleaned. To capture this, we need the following modifications of the existing domain:

- We need to able to express that a brush is clean. For this we will use a unary predicate *Clean*.
- We need to be able to clean an unclean brush. For this we will introduce a new action schema *CleanBrush*.
- We will require that an unclean brush can never be dipped into another color than the one it previously had. Hence, we need to separate between dipping a clean and an unclean brush, as these will have different preconditions. This is most easily done by having two separate action schemas, *DipCleanBrush* and *DipBrush*. The first action is only applicable when the brush is clean, but allows us to dip the brush in any can of any color. The second is only applicable when the color of the brush is the same as the color in the can.
- So far we have used a literal $CanPaint(r, k)$ to express that brush r can paint in color k. Now things are slightly more complicated. After a brush has been used to paint, it is dry and can't paint before being dipped again—however, it still has a bit of color left, and can only be dipped in the same color unless being cleaned first. This means we need to be able to assign colors to dry brushes as well. We can do this by splitting $CanPaint(r, k)$ into two atoms $ColorOfBrush(r, k)$ and $CanPaint(r)$. Then $ColorOfBrush(r, k)$ means that brush r has color k, and $CanPaint(r)$ means that the brush is wet (it has been dipped, but afterwards neither cleaned nor used to paint). A dipped brush should then preserve its color even after having been used to paint, however it will no longer be wet. The brush color only disappears when the brush is cleaned.

Fill in the preconditions and effects of the action schemas of Figure 2 such that the resulting schemas follow the specifications provided above. Make sure to use a *minimal number of literals*, that is, never add a precondition or effect literal that is not necessary to include. Superfluous literals will be counted as mistakes. In the figure, each precondition and effect has 4 fillable text fields, one for each literal. If you need fewer than 4, just leave the remaining text fields blank.

ACTION : $DipCleanBrush(r, c, k)$ // dip clean brush r in can c containing paint of color k

PRECONDITION : Clean(r) ∧ ColorInCan(c,k) ∧ ∧

EFFECT : ColorOfBrush(r,k) ∧ CanPaint(r) ∧ -Clean(r) ∧

ACTION : $DipBrush(r, c, k)$ // dip brush r in can c containing paint of color k

PRECONDITION : -Clean(r) ∧ ColorOfBrush(r,k) ∧ ColorInCan(c,k) ∧

EFFECT : CanPaint(r) ∧ ∧ ∧

ACTION : $Paint(b, r, k)$ // paint block b in color k using brush r

PRECONDITION : ColorOfBrush(r,k) ∧ CanPaint(r) ∧ ∧

EFFECT : ColorOfBlock(b,k) ∧ -CanPaint(r) ∧ ∧

ACTION : $CleanBrush(r, k)$ // clean brush r having color k

PRECONDITION : ColorOfBrush(r,k) ∧ ∧ ∧

EFFECT : -ColorOfBrush(r,k) ∧ Clean(r) ∧ ∧

Figure 2: Action schemas for the modified domain

You can assume that the initial state and goal of any problem in the modified domain is as for the original domain, except we suppose all brushes to be initially clean, i.e. we add $Clean(r)$ for each brush r to the initial state description.

Q8 Optimal plans in modified domain

Compute an optimal plan using the action schemas from the previous question in the planning problem having:

- Two brushes $R1$ and $R2$ (initially clean, by convention for initial states).
- A can $C1$ with paint of color G (green), a can $C2$ with paint of color R (red) and a can $C3$ with paint of color B (blue).
- Four blocks $B1, \ldots, B4$.
- The goal is for blocks $B1$ and $B2$ to be green, block $B3$ to be red and block $B4$ to be blue.

What is the length of the computed optimal plan? 9