02285 AI and MAS, F24
Theory Assignment
Due: 11 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.

Handing in. This assignment is to be handed in as the "Theory Assignment" 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. A file `notes.pdf` with all the calculations, graphs, reasoning and intermediate results that you produced in order to complete the assignment. No particular format is required here. It's recommended to do handwritten notes and then scan them afterwards.

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 Adobe Acrobat or Mac Preview that supports fillable forms. We will extract and check the answers automatically, so it's extremely important to follow all 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. As the answers are automatically corrected, it's essential that you avoid spelling mistakes! 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 (tilde), for instance `~ColorOfBlock(B1,G)`.

When you do the exercises, make sure to describe all your calculations and reasoning on a separate piece of paper (the one you hand in as `notes.pdf`). Many of the questions refer back to previous questions, so you will need to be able to go back and look at the details of your answers to those questions. Furthermore, many of the questions require a good deal of thinking and computation, and will be almost impossible to answer correctly unless you make good and well-structured notes on the side.

Exercise 1 (Hospital domain: Action schemas, state spaces, IW algorithm, multi-agent conflicts)

In this exercise we consider how to represent the hospital domain in PDDL. At first we consider only the single-agent case. 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:

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

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.

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. Leave remaining text fields blank.

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

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

EFFECT : AgentAt(0,agtto) ∧ Free(agtfrom) ∧ ∧
∧ ∧ ∧
∧ ∧

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

PRECONDITION : Free(boxto) ∧ Neighbour(agtfrom, boxfrom) ∧ Type(box,0) ∧
Neighbour(boxfrom, boxto) ∧ ∧
∧ ∧

EFFECT : AgentAt(0,boxfrom) ∧ BoxAt(box, boxto) ∧ Free(agtfrom) ∧
∧ ∧ ∧
∧ ∧

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

PRECONDITION : Free(agtto) ∧ Neighbour(agtfrom, boxfrom) ∧ Type(box,0) ∧
Neighbour(agtto, agtfrom) ∧ ∧
∧ ∧

EFFECT : AgentAt(0,agtto) ∧ BoxAt(box, agtfrom) ∧ Free(boxfrom) ∧
∧ ∧ ∧
∧ ∧

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

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

Initial state:

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

After *Move* action:

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

After *Pull* action:

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

After *Push* action:

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

Figure 3: State sequence of plan.

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:

$$Move(0, L3, L2), Pull(0, L2, L3, B1, L4), Push(0, L3, B1, L2, L1).$$

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 always, leave unused text fields blank.

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.

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

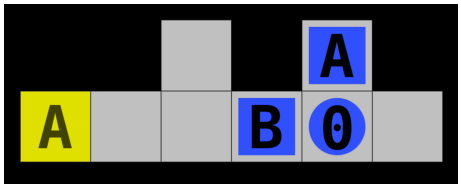

Figure 4: A single-agent hospital level.

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 13
2. The length of the solution found by $IW(1)$ is (write `infinity` if no solution was found) infinity
3. No matter in which order we add child states to the frontier, $IW(1)$ will necessarily find the optimal solution. ☐
4. For some order of adding child states to the frontier, $IW(1)$ will not find a solution. ✔
5. For some order of adding child states to the frontier, $IW(1)$ will find an optimal solution. ✔
6. The problem has width at most 2. ✔
7. For some order of adding child states to the frontier, the effective width of the problem is 2. ✔
8. No matter in which order we add child states to the frontier, $IW(2)$ will necessarily find the optimal solution. ☐
9. Any solvable single-agent planning problem in the hospital domain with a single agent and a single box has width at most 2. ✔
10. 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. ☐
11. Consider the level shown in Figure 4. No matter in which order we add child states to the frontier, $IW(2)$ will necessarily find the optimal solution. ☐

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:

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

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? 0
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? 0
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 $\mathit{Move}(0, \dots)$, $\mathit{Push}(0, \dots)$ or $\mathit{Pull}(0, \dots)$. 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 $\mathit{CellConflict}(a_i, a_j)$ holds then the effects of a_i and a_j are mutually inconsistent. ☐
2. If $\mathit{CellConflict}(a_i, a_j)$ holds then the precondition of one of the actions is inconsistent with the effect of the other. ☐
3. If $\mathit{BoxConflict}(a_i, a_j)$ holds then the effects a_i and a_j are mutually inconsistent. ☐
4. If $\mathit{BoxConflict}(a_i, a_j)$ then the precondition of one of the actions is inconsistent with the effect of the other. ☐
5. For all applicable actions a_i and a_j where the precondition of one is inconsistent with the effect of the other, either $\mathit{CellConflict}(a_i, a_j)$ or $\mathit{BoxConflict}(a_i, a_j)$ holds. ☐
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). ☐

Exercise 2 (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 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 $ColorOfBlock(b, k)$ atoms are true in the initial state, hence all blocks are initially uncolored. Also, none of the $CanPaint(r, k)$ atoms are true in the initial state, so initially none of the brushes can paint in any color.

Q6 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
- [ ] $CanPaint$
- [x] $ColorInCan$
- [ ] $ColorOfBlock$
- [x] $IsColor$

Q7 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 : [x] $Brush(r)$ ∧ [x] $Can(c)$ ∧ [ ] $IsColor(k)$ ∧ [x] $ColorInCan(c, k)$

EFFECT : $CanPaint(r, k)$

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

PRECONDITION : [x] $Block(b)$ ∧ [x] $Brush(r)$ ∧ [ ] $IsColor(k)$ ∧ [x] $CanPaint(r, k)$

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.

Q8 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 [7]
2. The total number of loop edges (edges with the same start and end point) is [4]

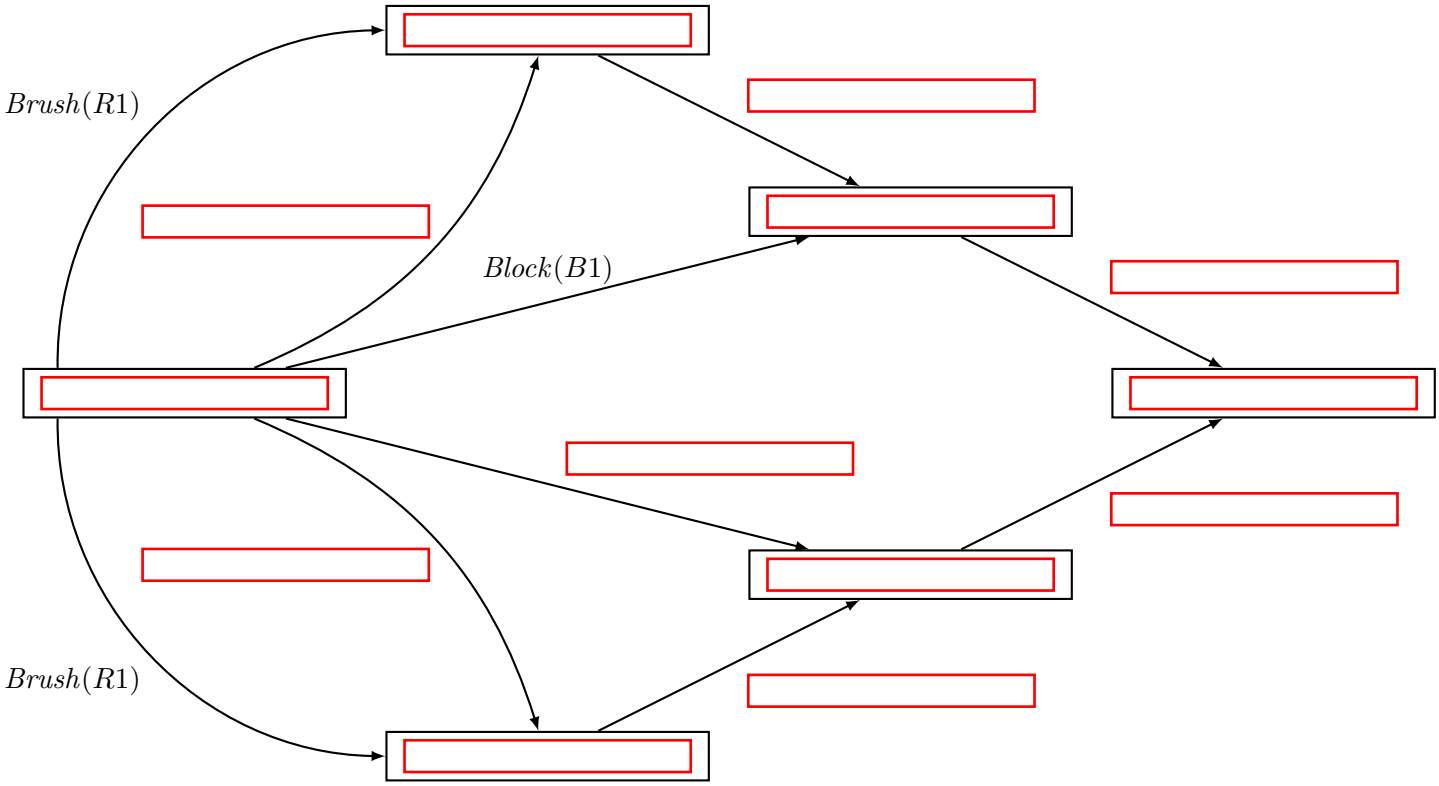

Figure 5: A partially ordered plan (POP graph).

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

Q9 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

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

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

Q10 POP algorithm

Consider the same planning problem $(\mathcal{A}, s_0, g)$ as above. The graph of Figure 5 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. 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?

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

Q11 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 Q9. Determine the following heuristic values:

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

Q12 Modifying the domain

Consider the following action sequence:

$$\mathit{DipBrush}(R1, C1, Y), \mathit{Paint}(B1, R1, Y),$$
$$\mathit{DipBrush}(R1, C2, B), \mathit{Paint}(B2, R1, B).$$

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*.

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

PRECONDITION : ☐ ∧ ☐ ∧ ☐ ∧ ☐

EFFECT : ☐ ∧ ☐ ∧ ☐ ∧ ☐

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

PRECONDITION : ☐ ∧ ☐ ∧ ☐ ∧ ☐

EFFECT : ☐ ∧ ☐ ∧ ☐ ∧ ☐

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

PRECONDITION : ☐ ∧ ☐ ∧ ☐ ∧ ☐

EFFECT : ☐ ∧ ☐ ∧ ☐ ∧ ☐

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

PRECONDITION : ☐ ∧ ☐ ∧ ☐ ∧ ☐

EFFECT : ☐ ∧ ☐ ∧ ☐ ∧ ☐

Figure 6: Action schemas for the modified domain

- 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 6 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.

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.

Figure 7: An optimal plan for the planning problem of Q13.

Q13 Optimal plans in modified domain

Provide 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.

Write the plan in Figure 7, one action per fillable text field. You might not need all text fields.