Knowledge, Reasonging, and Planning

This assignment is **individual**.

Deadline: The deadline is **Oct 17, 8pm** and it is strict.

Submission: Please, submit your solutions in Canvas. Submit your solution to the first exercise as a single .zip file under **A3.PP PDDL zip submission** in Assignments. Submit your solutions to the rest of the exercises as a single .pdf file under **A3.PP pdf submission**. Scanned handwriting is OK as long as it is clearly readable.

Grade E: You need to receive 45/50 points for the E-level exercises, i.e. from 1.1, 2.1, 2.2, 3.1, 3.2, and 3.3.

Grade D: You need to receive 72/80 points for the E-level and D-level exercises, i.e. from 1.1, 1.2, 2.1, 2.2, 2.3, 3.1, 3.2, 3.3, and 3.4.

Grade C: You need to receive 108/120 points for the E-level, D-level, and C-level exercises, i.e. from 1.1, 1.2, 2.1, 2.2, 2.3, 2.4, 2.5, 3.1, 3.2, 3.3, 3.4, 3.5, and 4.1.

Grade B: You need to receive 144/160 points for all the exercises.

1 Beekeeper

In this exercise, we are going to look into encoding the following domain using PDDL:

There is a number of bee houses at different positions. A beekeeper can do the following:

- **Move**: She is able to move from one position to another.
- **Wear**: She is able to put on a costume.
- **Take off:** She is able to take off the costume.
- **Collect:** She is able to collect all honey from a bee house and fill a honey pot. She has to be wearing a costume and have a free arm to do that.
- **Pick up:** She is able to pick up a honey pot.
- **Drop off:** She is able to drop off the honey pot.

1.1 E-level 10 points

Download the package **Beekeeper1.zip** from Canvas. Your task is to complete the file **honey-domain.pddl** to formalize this domain. The syntax used in the file is a standardized syntax used in state-of-the-art PDDL solvers, such as in this on-line editor and solver [2]. There are numerous examples of problems encoded in this syntax under the Import tab in this tool. There are also numerous tutorials on this syntax, for instance this one [1]. The relevant tab to explore there is PDDL Background.

Note that the file honey-domain.pddl will only contain the definition of the *domain*. *Problem instances* including the definition of objects in the world, the initial state and the goal specification are given in separate files. You can find three examples of problem instances for the domain in this exercise in the **Honey-domain-problems** subfolder. They are named **problem-i.pddl**, where i is the number of the problem.

Your task is to complete only the code for each of the six actions, which involves writing the parameters, the precondition and the effect. Note that each action comes with a comment that gives more details that the brief domain introduction above. All the predicates you are allowed to use are already given in the file. You will not need to define any requirements or functions.

You can use the above mentioned on-line editor and solver [2] to see whether your domain definition allows to find a solution to problem-i.pddl.

Alternatively, we provided **solve.py** that calls a PDDL solver. Its usage is the following:

```
$ cd <path-to-folder>
$ python solve.py <path-to-domain> <path-to-problem> <path-to-solution> [-v]
```

If the *-v* option is selected then the path obtained and its cost will be shown in your terminal. NOTE: The **solve.py** shall be executed with Python 2.7.

1.2 D-level 10 points

Download the package **Beekeeper2.zip** from Canvas. Your second task is to extend the domain definition honey-domain.pddl to **honey-domain-with-distance.pddl** to be able to handle the goal of minimizing total distance the beekeeper travels and complete the definition of a problem **problem-with-distance.pddl** that you find in the package. The problem is illustrated in Figure 1.1. Similarly as in the three pddl problem definitions you were given before, the honey pots are initially empty and there is honey in the bee houses. The goal is to have both honey pots full at the house and have the costume returned in the garden. Unlike in the previous three pddl

problems, it is not possible to travel from any position to any other position. The edges show paths between positions and the beekeeper can only travel on these paths. The weights of the edges give the respective distances.

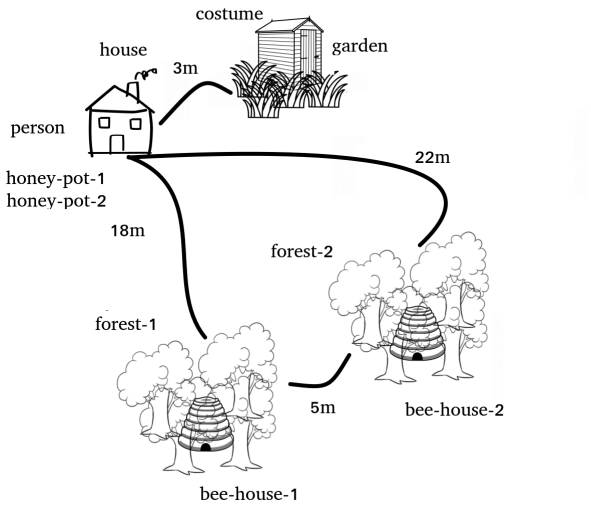

Figure 1.1: Illustration of a problem with distances to be defined in honey-domain-with-distance.pddl.

The tutorial [1] we already mentioned above is a good starting point to learn how to add numeric expresions to your PDDL (or, more precisely now PDDL2.1) code. The relevant tabs to explore there are Numeric Expressions, Conditions, and Effects, and also Plan Metrics.

A few hints:

- You will need to encode the distances of the paths in the problem definition problem-with-distance.pddl with the use of numeric *fluents*. You will also need to encode that there is no path in between some of the positions and you can achieve that by assigning a negative distance to that path.
- Since you will need to work with numeric fluents, you will need to add (:requirements :fluents) to the domain definition honey-domain-with-distance.pddl. You should not need any other requirements.
- You will need to introduce *functions* in the domain definition to keep track of the distance traveled and to be able to work with distances between two positions.
- Some of the action schemes in the domain definition will now change, too: sometimes, you will need to check whether there is a path between two places and sometimes, you will need to utilize *increase* in order to be able to keep track of the distances traveled so far.

How to submit: Under the assignment A3.PP.PDDL, wrap into .zim that contains honey-domain.pddl, honey-domain-with-distance.pddl, and problem-with-distance.pddl

References

[1] A PDDL 2.1 tutorial, `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/fox03a-html/JAIRpddl.html`, Accessed: 2018-09-26

[2] An online PDDL editor and solver, `http://editor.planning.domains/`, Accessed: 2018-09-26

2 Einstein Puzzle with a Twist

Bob, Carol, Ted, and Alice are a group of people who live in two neighboring houses. Pairwise, they are either neighbors or roommates. One of the houses is red and one of houses is blue. Bob, Carol, Ted, and Alice are each a huge fan of one band: for each of them it is either Abba, or Beatles. They each also favor one drink: either beer, snaps, or milk. And they each also are afraid either of spiders, or of elevators. Furthermore, we know the following:

S1: Bob lives in the red house.

S2: Alice favors snaps.

S3: Ted is afraid of elevators.

S4: Carol is Ted's roommate.

S5: Everyone who is afraid of elevators lives in the blue house.

S6: Everyone who lives in the red house is afraid of spiders.

S7: Everyone who favors snaps lives in the red house.

S8: All neighbors of Bob favor beer.

S9: All roommates of all people who are afraid of elevators are afraid of spiders.

S10: All neighbors of all people who favor milk are fans of Beatles.

S11: All roommates of Alice that are afraid of spiders favor milk.

S12: All roommates of all people who favor snaps are fans of Beatles.

S13: All neighbors of Ted who favor snaps and are afraid of spiders are fans of Abba.

2.1 E-level 10 points

Write down the first-order logic representations of the above 13 premises, suitable for use with Generalized Modus Ponens. Use constants

$$\mathit{Bob}, \mathit{Carol}, \mathit{Ted}, \mathit{Alice}, \mathit{Red}, \mathit{Blue}, \mathit{Abba}, \mathit{Beatles}, \mathit{Beer}, \mathit{Snaps}, \mathit{Milk}, \mathit{Spiders}, \mathit{Elevators};$$

functions

$$\mathit{House}(x), \mathit{Drink}(x), \mathit{Fear}(x), \mathit{Music}(x);$$

and predicates

$$\mathit{Roommate}(x, y), \mathit{Neighbor}(x, y),$$

meaning that x is a roommate of y, and x is a neighbor of y, respectively. A hint: you will not need existential quantifier.

For instance, premises $S1$, $S6$ and $S8$ can be written as follows:

S1: $\mathit{House}(\mathit{Bob}) = \mathit{Red}.$

S6: $\forall x.\mathit{House}(x) = \mathit{Red} \implies \mathit{Fear}(x) = \mathit{Spiders}.$

S9: $\forall x, y.\mathit{Roommate}(x, y) \wedge \mathit{Fear}(y) = \mathit{Elevators} \implies \mathit{Fear}(x) = \mathit{Spiders}.$

2.2 E-level 5 points

Infer sentence $Fear(Carol) = Spiders$ using Generalized Modus Ponens from the knowledge base including only the above 13 premises S1 - S13. Recall that Generalized Modus Ponens is an inference rule

$$\frac{p'_1, p'_2, \ldots, p'_n,\ (p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow q)}{SUBST(\theta, q)},$$

where θ is a substitution such that $SUBST(\theta, p'_i) = SUBST(\theta, p_i)$, and p_i, p'_i are atomic sentences, for all i.

For instance, take premises S1 and S6. p'_1 is $House(Bob) = Red$, p_1 is $House(x) = Red$, q is $Fear(x) = Spiders$, and $\theta = \{x/Bob\}$. Notice that $SUBST(\theta, p'_1) = SUBST(\theta, p_1)$, which is $House(Bob) = Red$, and $SUBST(\theta, q)$ is $Fear(Bob) = Spiders$.

Write down each application of Generalized Modus Ponens following this form, including θ:

$$\frac{S1 : House(Bob) = Red \quad S6 : (House(x) = Red \Rightarrow Fear(x) = Spiders)}{S14 : Fear(Bob) = Spiders}, \text{ where } \theta = \{x/Bob\}.$$

2.3 D-level 10 points

Convert each premise S1-S13 to the conjunctive normal form (CNF). In other words, convert each of them to a clause, where clause is a disjuction of literals. Following the notation from the slides and the course book, literals are atomic sentences or their negations. Denote the 13 clauses you obtain C1 - C13.

2.4 C-level 15 points

Consider the knowledge base containing C1 - C13 you just obtained and four additional sentences in CNF:

C14: $\neg Roommates(x, y) \vee \neg House(x) = z \vee House(y) = z$.

C15: $\neg House(x) = z \vee \neg House(y) = z \vee Roommates(x, y)$.

C16: $\neg Roommates(x, y) \vee Roommates(y, x)$.

C17: $\neg House(x) = Red \vee \neg House(y) = Blue \vee Neighbors(x, y)$.

C18: $\neg Neighbors(x, y) \vee Neighbors(y, x)$.

- Express sentences C14 - C18 in good, natural English. No x's and no y's.
- Infer who is a fan of which band using Resolution from the knowledge base including only the premises C1 - C18. Recall that resolution rule is

  $$\frac{\ell_1 \vee \ldots \vee \ell_k,\ m_1 \vee \ldots \vee m_n}{SUBST(\theta, \ell_1 \vee \ldots \ell_{i-1} \vee \ell_{i+1} \ldots \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)},$$

  where $UNIFY(\ell_i, \neg m_j) = \theta$ with the property that θ satisfies $SUBST(\theta, l_i) = SUBST(\theta, \neg m_j)$.

  For instance, take clauses $C1 : House(Bob) = Red$ and $C6 : \neg House(x) = Red \vee Fear(x) = Spiders$. ℓ_1 is $House(Bob) = Red$, m_1 is $\neg House(x) = Red$, m_2 is $Fear(x) = Spiders$, and $UNIFY(\ell_1, \neg m_1) = \theta = \{x/Bob\}$. The resolvent clause is $Fear(Bob) = Spiders$. Write down each resolution following this form, including the unifier θ:

  $$\frac{C1 : House(Bob) = Red, \quad C6 : \neg House(x) = Red \vee Fear(x) = Spider}{C19 : Fear(Bob) = Spiders}, \text{ where } \theta = \{x/Bob\}. \tag{2.1}$$

2.5 C-level 5 points

Construct an example of two clauses that can be resolved together in two different ways giving two different outcomes. Show the two different resolutiosns.

3 Treasure Hunt

A captain on a boat hunts for a coin from a treasure. From the bay, she can sail to white waters, from there to blue waters, from there to green waters, and she can keep sailing in green waters forever, or return to blue waters, then to white waters, and all the way to the bay. She can for instance also sail to blue waters, turn there and return to the bay. There is a treasure at the bottom of the sea in the blue waters. To get a coin from the treasure, the captain needs to dive, pick a coin, and surface. The story can be formalized in PDDL:

$$
\begin{aligned}
&\mathit{Init}\ \big(\mathit{Next(Bay,White)} \wedge \mathit{Next(White,Blue)} \wedge \mathit{Next(Blue,Green)} \wedge \mathit{Next(Green,Green)} \wedge \\
&\qquad \mathit{CaptainOnBoard} \wedge \mathit{Boat(Bay)} \wedge \mathit{TreasureAtBottom(Blue)}\big) \\
&\mathit{Goal}\ \big(\mathit{Boat(Bay)} \wedge \mathit{CaptainOnBoard} \wedge \mathit{CaptainHasCoin}\big) \\
&\mathit{Action}\ \big(\mathit{SailOut}(\mathit{from}, \mathit{to}) \\
&\qquad \text{PRECOND}: \mathit{Next}(\mathit{from}, \mathit{to}) \wedge \mathit{Boat}(\mathit{from}) \wedge \mathit{CaptainOnBoard} \\
&\qquad \text{EFFECT}: \neg \mathit{Boat}(\mathit{from}) \wedge \mathit{Boat}(\mathit{to})\big) \\
&\mathit{Action}\ \big(\mathit{SailIn}(\mathit{from}, \mathit{to}) \\
&\qquad \text{PRECOND}: \mathit{Next}(\mathit{to}, \mathit{from}) \wedge \mathit{Boat}(\mathit{from}) \wedge \mathit{CaptainOnBoard} \\
&\qquad \text{EFFECT}: \neg \mathit{Boat}(\mathit{from}) \wedge \mathit{Boat}(\mathit{to})\big) \\
&\mathit{Action}\ \big(\mathit{Dive} \\
&\qquad \text{PRECOND}: \mathit{CaptainOnBoard} \\
&\qquad \text{EFFECT}: \neg \mathit{CaptainOnBoard} \wedge \mathit{CaptainAtBottom}\big) \\
&\mathit{Action}\ \big(\mathit{Pick}(\mathit{at}) \\
&\qquad \text{PRECOND}: \mathit{Boat}(\mathit{at}) \wedge \mathit{CaptainAtBottom} \wedge \mathit{TreasureAtBottom}(\mathit{at}) \\
&\qquad \text{EFFECT}: \mathit{CaptainHasCoin}\big) \\
&\mathit{Action}\ \big(\mathit{Surface} \\
&\qquad \text{PRECOND}: \mathit{CaptainAtBottom} \\
&\qquad \text{EFFECT}: \neg \mathit{CaptainAtBottom} \wedge \mathit{CaptainOnBoard}\big)
\end{aligned}
$$

3.1 E-level 10 points

We will look into converting the planning problem to planning in a state space. Finish drawing the space of all states that are reachable from the initial one in Figure 3.1. Represent each state as the list of fluents (ground, functionless atoms) that are true therein. In classical planning, we make the *closed-world assumption* meaning that any fluent not mentioned in the state is false. However, because $\mathit{Next}(\mathit{Bay}, \mathit{White})$, $\mathit{Next}(\mathit{White}, \mathit{Blue})$, $\mathit{Next}(\mathit{Blue}, \mathit{Green})$, $\mathit{Next}(\mathit{Green}, \mathit{Green})$, and $\mathit{TreasureAtBottom}(\mathit{Blue})$ are always true, for simplicity of presentation, you do not have to list them explicitly as labels in each state. Label each transition in the state space with the corresponding ground action.

3.2 E-level 5 points

Answer the following questions and *provide the motivation* for your answers:

- How many different plans (sequences of actions) that lead to the satisfaction of the goal are there?

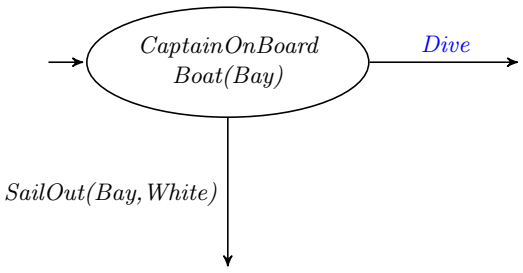

Figure 3.1: A part of the state space.

- Assume that the captain cannot keep sailing in green waters forever, that from green waters she can only sail back to blue waters. In other words, the initial state changed as follows:

$$Init\,\big(Next(Bay,White)\,\wedge\,Next(White,Blue)\,\wedge\,Next(Blue,Green)\,\wedge\\ CaptainOnBoard\,\wedge\,Boat(Bay)\,\wedge\,TreasureAtBottom(Blue)\big)$$

  How many different plans (sequences of actions) that lead to the satisfaction of the goal are there now?

- If there are multiple plans, what is the optimal plan in terms of the number of actions? Give the plan as a sequence of ground actions.

3.3 E-level 10 points

Consider a variant of the treasure hunt problem, where the captain was blinded and does not know whether the boat is in the bay, or in white, blue, or green waters. This means that the environment is partially observable, we are facing sensorless planning, and we need to work with the belief-state space instead. Instead of representing a belief state as an explicit enumeration of the set of states the world can be in, we can use use logical formulas. We make the *open-world assumption* meaning that representation of a belief state can contain both positive and negative fluents, and if a fluent does not appear in the belief state, the fluent is unknown.

Draw the space of all belief states that are reachable from the initial one, assuming the same action schema as above. Similarly as above, for simplicity of the presentation, you do not have to explicitly list the fluents that are either always true or always false. Label each transition in the belief state space with the corresponding ground action. Notice, that an action can be applied in a belief state only if its preconditions are satisfied by every state in that belief state.

TIP: We say that you should label transitions with ground actions. This means that both variables from and to in SailOut(from,to) have to be substituted with constants. If you do that, can you actually make sure that the precondition of that action is satisfied in every single state in your belief state?

Answer the following questions and *provide the motivation* for your answers:

- Give 3 examples of fluents that are always true and 3 examples of fluents that are always false in the given environment.

- How many actual physical states does the initial belief state contain?

- How many different plans that lead to the satisfaction of the goal are there?

- If there are multiple plans, what is the optimal plan in terms of the number of actions? Give the plan as a sequence of ground actions.

3.4 D-level 10 points

Consider now that the captain can sail to anywhere from anywhere, represented by replacing SailIn and SailOut with these alternative action schemas:

$$
\begin{aligned}
&\mathit{Action}\ \big(\mathit{SailToBay}\\
&\qquad \text{PRECOND}: \mathit{CaptainOnBoard}\\
&\qquad \text{EFFECT}: \mathit{Boat}(\mathit{Bay}) \wedge \neg \mathit{Boat}(\mathit{White}) \wedge \neg \mathit{Boat}(\mathit{Blue}) \wedge \neg \mathit{Boat}(\mathit{Green})\big)\\
&\mathit{Action}\ \big(\mathit{SailToWhite}\\
&\qquad \text{PRECOND}: \mathit{CaptainOnBoard}\\
&\qquad \text{EFFECT}: \mathit{Boat}(\mathit{White}) \wedge \neg \mathit{Boat}(\mathit{Bay}) \wedge \neg \mathit{Boat}(\mathit{Blue}) \wedge \neg \mathit{Boat}(\mathit{Green})\big)\\
&\mathit{Action}\ \big(\mathit{SailToBlue}\\
&\qquad \text{PRECOND}: \mathit{CaptainOnBoard}\\
&\qquad \text{EFFECT}: \mathit{Boat}(\mathit{Blue}) \wedge \neg \mathit{Boat}(\mathit{Bay}) \wedge \neg \mathit{Boat}(\mathit{White}) \wedge \neg \mathit{Boat}(\mathit{Green})\big)\\
&\mathit{Action}\ \big(\mathit{SailToGreen}\\
&\qquad \text{PRECOND}: \mathit{CaptainOnBoard}\\
&\qquad \text{EFFECT}: \mathit{Boat}(\mathit{Green}) \wedge \neg \mathit{Boat}(\mathit{Bay}) \wedge \neg \mathit{Boat}(\mathit{White}) \wedge \neg \mathit{Boat}(\mathit{Blue})\big)
\end{aligned}
$$

Answer the questions from Sec. 3.2 and Sec. 3.3 and *motivate your answers*, i.e.

- Draw the belief state space assuming that the captain initially does not know whether the boat is in the bay, or in the white, blue, or green waters. Again, for simplicity of presentation, you do not have to explicitly list the fluents that are either always true or always false.
- How many actual physical states does the initial belief state contain?
- How many different plans that lead to the satisfaction of the goal are there?
- If there are multiple plans, what is the optimal plan in terms of the number of actions? Give the plan as a sequence of ground actions.

3.5 C-level 10 points

Consider now yet another variant of the treasure hunt problem, where the captain again knows where the boat is, but does not know, where the treasure is. However, when she is at the bottom of the sea, she can perceive whether the treasure is there or not. The problem is not sensorless anymore. Formally, we add the following to the domain description:

$$
\begin{aligned}
&\mathit{Percept}\ \big(\mathit{TreasureAtBottom}(at)\\
&\qquad \text{PRECOND}: \mathit{CaptainAtBottom} \wedge \mathit{Boat}(at)\big)
\end{aligned}
$$

- Draw **a connected subgraph of** the belief state space **containing at least 10 different belief states** assuming that the captain initially does not know where the treasure is. Use the action schemas from the beginning of Sec. 3 and again, for simplicity of presentation, you do not have to explicitly list the fluents that are either always true or always false. Each transition in your belief state space should be labeled either with ground action or with perceived fluent in case *Percept* was made. For instance, from the belief state, where the captain knows that she is at the bottom of the sea and that the boat is in the bay, according to the domain specification, she can surface or perceive whether there is a treasure or not. An example of the transitions labels from this state are given in Fig. 3.2.

TIP: There are two important assumptions that we make in this exercise: there is a treasure somewhere and there is only one treasure. Now practically that means that belief states that have for instance TreasureAtBottom(Blue) true and TreasureAtBottom(Bay) true are not feasible. It also means that if Percept returns that TreasureAtBottom(Bay) is true, you also know that TreasureAtBottom(Blue) is false. It also means that once TreasureAtBottom(Bay) is false, making Percept will always return that it is false. So this way, when drawing only the feasible states, you can prune the belief state space significantly.

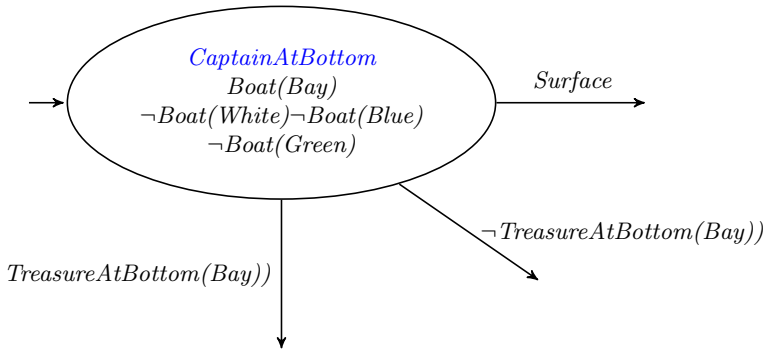

Figure 3.2: Possible transitions from one of the belief states.

- You may notice that there does not exist a sequence of actions that leads to satisfying the goal. However, there exists a *contingent plan* doing so. Find such a contingent plan and represent it as a sequence of actions and if-then-else structures. An example of a contingent plan (that however does not satisfy the goal) is

$$
\begin{array}{l}
[\,Dive,\\
\quad \textbf{if}\, Treasure(Bay)\\
\quad\quad \textbf{then}\,[\,Pick(Bay), Surface\,]\\
\quad\quad \textbf{else}\,[Surface,\ Dive,\ \textbf{if}\ Treasure(Bay)\\
\qquad\qquad\qquad\qquad\qquad \textbf{then}\,[\,Pick(Bay), Surface\,]\\
\qquad\qquad\qquad\qquad\qquad \textbf{else}\,[NoOp]]].
\end{array}
$$

  NoOp is used to denote that no action is going to be taken.

  TIP: Note that with the two assumptions – that there is a treasure and that there is only one treasure in our world – you will be able to find a contingent plan, but without assuming that there is a treasure somewhere, you won't.

3.6 B-level 15 points

Assume an arbitrary PDDL planning domain, not necessarily one with a captain and boat. We will examine two general cases. The first case is fully observable, meaning that for each fluent we always know whether it is true or false. The second case is sensorless, meaning that at any moment, the value of each fluent can be true, false, or unknown. Suppose that in this second case, in the initial state, the value of some of the fluents are unknown.

- How does the size of the state space in the first fully observable case compare in general to the size of the belief state space in the second sensorless case? Is the state space always larger, always smaller or sometimes larger in size and sometimes smaller than the belief state space? If always larger or always smaller, give a proof. Otherwise, give examples of domains and initial states, for which the state space is smaller, respectively larger than the belief state space.

- Compare the size of the representation of the inital state in the first fully observable case and the initial belief state in the second sensorless case. Count the size of the representation as the number of fluents listed in the state and assume that the fluents, whose value is always true or always false are not explicitly listed. Similarly as above, if the state is always larger in size or always smaller than the belief state, give a proof. Otherwise, give examples of domains and initial states, for which the state is smaller, respectively larger than the belief state space.

4 Exercise 4

The following PDDL describes a robot in a 10x10 grid-like world that can move, pick objects, and drop them. The robot can hold only one object at a time and its goal is to gather all objects in cell $C_{1,1}$.

$$
\begin{aligned}
&\mathit{Init}\ \big(\mathit{Free}(\mathit{Robot})\ \wedge\ \mathit{At}(\mathit{Robot}, C_{1,1}) \wedge \\
&\quad \mathit{At}(\mathit{Ball}, C_{4,6}) \wedge \mathit{At}(\mathit{Cube}, C_{3,7}) \wedge \mathit{At}(\mathit{Basket}, C_{10,2}) \wedge \\
&\quad \mathit{Object}(\mathit{Ball}) \wedge \mathit{Object}(\mathit{Cube}) \wedge \mathit{Object}(\mathit{Basket}) \wedge \\
&\quad \mathit{Next}(C_{1,1}, C_{1,2}) \wedge \mathit{Next}(C_{1,1}, C_{2,1}) \wedge \\
&\quad \mathit{Next}(C_{1,2}, C_{1,1}) \wedge \mathit{Next}(C_{1,2}, C_{2,2}) \wedge \mathit{Next}(C_{1,2}, C_{1,3}) \wedge \ldots \wedge \\
&\quad \mathit{Next}(C_{1,10}, C_{1,9}) \wedge \mathit{Next}(C_{1,10}, C_{2,10}) \wedge \ldots \wedge \\
&\quad \mathit{Next}(C_{10,10}, C_{10,9}) \wedge \mathit{Next}(C_{10,10}, C_{9,10})\big) \\
&\mathit{Goal}\ \big(\mathit{At}(\mathit{Robot}, C_{1,1})\ \wedge\ \mathit{At}(\mathit{Ball}, C_{1,1})\ \wedge\ \mathit{At}(\mathit{Cube}, C_{1,1})\ \wedge\ \mathit{At}(\mathit{Basket}, C_{1,1})\big) \\
&\mathit{Action}\ \big(\mathit{Move}(\mathit{from}, \mathit{to}) \\
&\quad \text{PRECOND} : \mathit{Next}(\mathit{from}, \mathit{to}) \wedge \mathit{At}(\mathit{Robot}, \mathit{from}) \\
&\quad \text{EFFECT} : \neg \mathit{At}(\mathit{Robot}, \mathit{from}) \wedge \mathit{At}(\mathit{Robot}, \mathit{to})\big) \\
&\mathit{Action}\ \big(\mathit{Pick}(\mathit{thing}, \mathit{place}) \\
&\quad \text{PRECOND} : \mathit{At}(\mathit{Robot}, \mathit{place}) \wedge \mathit{At}(\mathit{thing}, \mathit{place}) \wedge \mathit{Object}(\mathit{thing}) \wedge \mathit{Free}(\mathit{Robot}) \\
&\quad \text{EFFECT} : \neg \mathit{At}(\mathit{thing}, \mathit{place}) \wedge \neg \mathit{Free}(\mathit{Robot}) \wedge \mathit{Holds}(\mathit{Robot}, \mathit{thing})) \\
&\mathit{Action}\ \big(\mathit{Drop}(\mathit{thing}, \mathit{place}) \\
&\quad \text{PRECOND} : \mathit{At}(\mathit{Robot}, \mathit{place}) \wedge \mathit{Holds}(\mathit{Robot}, \mathit{thing}) \wedge \mathit{Object}(\mathit{thing}) \\
&\quad \text{EFFECT} : \mathit{At}(\mathit{thing}, \mathit{place}) \wedge \mathit{Free}(\mathit{Robot}) \wedge \neg \mathit{Holds}(\mathit{Robot}, \mathit{thing}))\big)
\end{aligned}
$$

4.1 C-level 10 points

We will look into and compare different domain-independent heuristics guiding forward search when the PDDL planning problem is translated into a state space search problem. Denote the state space (S, T), where S is the set of states and T is a set of transitions (edges) between the states. Consider:

H1 Ignoring all preconditions of all actions in the action scheme.

H2 Ignoring precondition $\mathit{Free}(\mathit{Robot})$ in the $\mathit{Pick}(\mathit{thing}, \mathit{place})$ action scheme

H3 Ignoring all delete lists.

H4 Ignoring the delete list in the the $\mathit{Pick}(\mathit{thing}, \mathit{place})$ action scheme

Each of the heuristics defines a new PDDL planning domain and a new state space (S', T'). The minimal number of transitions between a state $s \in S'$ in this state space and a goal state is $h(s)$, and h can be used as a heuristics to guide search in (S, T). This means that $H1 - H4$ define heuristics $h_1 - h_4$ for search in the state space (S, T).

Answer the following questions and *provide motivation for each of the answers*:

- What is the value of h_1 for the initial state?

- What is the value of h_2 for the initial state?
- What is the value of h_3 for the initial state?
- What is the value of h_4 for the initial state?
- Compare heuristics h_1 and h_2. Does one of them dominate the other?
- Compare heuristics h_3 and h_4. Does one of them dominate the other?
- Compare heuristics h_2 and h_4. Does one of them dominate the other?

4.2 B-level 10 points

Overview paper [1] and focus on understanding the following terms:

- *goal distance,*
- *dead end,*
- *recognized dead end,*
- *unrecognized dead end.*

Answer the following questions for above robot in the grid problem and *provide motivation for each of the answers*:.

- Does there exist a reachable recognized dead end for H3?
- Does there exist a reachable unrecognized dead end for H3?

4.3 B-level 10 points

Let us now consider a modified version of the robot in the grid, where the world is split into a city and a dark forest. Once robot enters the forest, it can only get out with a flashlight.

$$
\begin{aligned}
\mathit{Init}\ \big(&\mathit{Free}(\mathit{Robot})\ \wedge\ \mathit{At}(\mathit{Robot}, C_{1,1}) \wedge \\
&\mathit{At}(\mathit{Ball}, C_{4,6}) \wedge \mathit{At}(\mathit{Cube}, C_{3,7}) \wedge \mathit{At}(\mathit{Basket}, C_{10,2}) \wedge \\
&\mathit{Object}(\mathit{Ball}) \wedge \mathit{Object}(\mathit{Cube}) \wedge \mathit{Object}(\mathit{Basket}) \wedge \\
&\mathit{Next}(C_{1,1}, C_{1,2}) \wedge \mathit{Next}(C_{1,1}, C_{2,1}) \wedge \\
&\mathit{Next}(C_{1,2}, C_{1,1}) \wedge \mathit{Next}(C_{1,2}, C_{2,2}) \wedge \mathit{Next}(C_{1,2}, C_{1,3}) \wedge \ldots \wedge \\
&\mathit{Next}(C_{1,10}, C_{1,9}) \wedge \mathit{Next}(C_{1,10}, C_{2,10}) \wedge \ldots \wedge \\
&\mathit{Next}(C_{10,10}, C_{10,9}) \wedge \mathit{Next}(C_{10,10}, C_{9,10}) \wedge \\
&\mathit{City}(C_{1,1}) \wedge \mathit{City}(C_{1,2}) \wedge \ldots \mathit{City}(C_{1,5}) \wedge \ldots \mathit{City}(C_{2,1}) \ldots \mathit{City}(C_{10,5}) \wedge \\
&\mathit{Forest}(C_{1,6}) \wedge \mathit{Forest}(C_{1,7}) \wedge \ldots \mathit{Forest}(C_{1,10}) \wedge \ldots \mathit{Forest}(C_{2,6}) \ldots \mathit{Forest}(C_{10,10})\big)
\end{aligned}
$$

$$
\mathit{Goal}\ \big(\mathit{At}(\mathit{Robot}, C_{1,1})\ \wedge\ \mathit{At}(\mathit{Ball}, C_{1,1})\ \wedge\ \mathit{At}(\mathit{Cube}, C_{1,1})\ \wedge\ \mathit{At}(\mathit{Basket}, C_{1,1})\big)
$$

$$
\begin{aligned}
\mathit{Action}\ \big(&\mathit{GrabFlashlight} \\
&\text{PRECOND}: \mathit{At}(\mathit{Robot}, C_{1,1}) \\
&\text{EFFECT}: \mathit{HasFlashlight}(\mathit{Robot})\big)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Action}\ \big(&\mathit{MoveInCity}(\mathit{from}, \mathit{to}) \\
&\text{PRECOND}: \mathit{City}(\mathit{from}) \wedge \mathit{Next}(\mathit{from}, \mathit{to}) \wedge \mathit{At}(\mathit{Robot}, \mathit{from}) \\
&\text{EFFECT}: \neg \mathit{At}(\mathit{Robot}, \mathit{from}) \wedge \mathit{At}(\mathit{Robot}, \mathit{to})\big)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Action}\ \big(&\mathit{MoveInForest}(\mathit{from}, \mathit{to}) \\
&\text{PRECOND}: \mathit{Forest}(\mathit{from}) \wedge \mathit{HasFlashlight}(\mathit{Robot}) \wedge \mathit{Next}(\mathit{from}, \mathit{to}) \wedge \mathit{At}(\mathit{Robot}, \mathit{from}) \\
&\text{EFFECT}: \neg \mathit{At}(\mathit{Robot}, \mathit{from}) \wedge \mathit{At}(\mathit{Robot}, \mathit{to})\big)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Action}\ \big(&\mathit{Pick}(\mathit{thing}, \mathit{place}) \\
&\text{PRECOND}: \mathit{At}(\mathit{Robot}, \mathit{place}) \wedge \mathit{At}(\mathit{thing}, \mathit{place}) \wedge \mathit{Object}(\mathit{thing}) \wedge \mathit{Free}(\mathit{Robot}) \\
&\text{EFFECT}: \neg \mathit{At}(\mathit{thing}, \mathit{place}) \wedge \neg \mathit{Free}(\mathit{Robot}) \wedge \mathit{Holds}(\mathit{Robot}, \mathit{thing}))
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Action}\ \big(&\mathit{Drop}(\mathit{thing}, \mathit{place}) \\
&\text{PRECOND}: \mathit{At}(\mathit{Robot}, \mathit{place}) \wedge \mathit{Holds}(\mathit{Robot}, \mathit{thing}) \wedge \mathit{Object}(\mathit{thing}) \\
&\text{EFFECT}: \mathit{At}(\mathit{thing}, \mathit{place}) \wedge \mathit{Free}(\mathit{Robot}) \wedge \neg \mathit{Holds}(\mathit{Robot}, \mathit{thing}))\big)
\end{aligned}
$$

Answer the following questions for this variant of the problem and *provide motivation for each of the answers*:

- Does there exist a reachable recognized dead end for H3?
- Does there exist a reachable unrecognized dead end for H3?

4.4 B-level 5 points

Suppose that you have a heuristics with the property that causes reachable unrecognized dead ends. Will that prevent finding a solution? Why?

References

[1] Jörg Hoffmann. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.