CHA2555 Assignment 1: Planning and Logic Programming

Artificial Intelligence

Hand-in date: 2018-01-26 16:00 GMT

1 Aims

To gain experience in representing and solving problems using symbolic AI methods in deterministic settings with full observability, and to critically reflect on this experience.

2 Overall Specification

This assignment is divided into two parts. In the first part, students will model several versions of dungeon-like games using the planning domain description language PDDL in order to test playability of levels. In the second part, students will represent variants of a simple game environment game using Answer Set Programming (ASP), which will allow enumeration of all possible scenarios for a given game state.

Deliverables will be encodings and sample problem instances for both parts with two accompanying short descriptive and reflective essays on the products.

The essays should be between 800 and 1000 words each and should be submitted by means of a single PDF document via turnitin on unilearn. The created files should be submitted as a zip file via unilearn. The submission is electronic only, no paper copies should be handed in.

Each assignment submission will receive a mark between 0 and 100 (each part amounting to a maximum of 50 marks), which will make up 50% of the overall module mark.

3 Detailed Specification

3.1 Part 1: Planning

3.1.1 Basic Domain

Consider the following domain: A single agent is inside a dungeon (or maze), of which it knows the exact layout. The dungeon consists of a number of rooms and each room can be connected to an arbitrary number of other rooms. If room A is connected to room B, then there is not necessarily a connection from B to A (this could be because of a door that opens only one way, or that the connection involves jumping downwards, but the exact reasons are not relevant to the assignment). Each room can also hold an arbitrary number of artifacts that can be examined. The agent can at any time be only in precisely one room, and there are two actions available to it: moving from the room it is currently in to a connected room; and collecting an artifact in the room it is currently in. Preconditions and effects of these actions should be straightforward.

In a problem instance, the rooms and connections of the dungeon and the artifacts and their locations are given, the initial position of the agent is specified and the goal consists of a number of artifacts to have been collected.

Represent this domain using PDDL, and provide 5 problem instances with 5 to 10 rooms and 3 to 5 artifacts. One of these problem instances should not admit any plan.

Solve all problem instances using a PDDL planner (one should admit no plan). The planner can be either from the itSIMPLE collection (in this case I must be able to import your PDDL into itSIMPLE and run the 5 planning problems with each planner and obtain solutions), or from the web. In the latter case you must provide the URL where you obtained the planners, their version numbers, and include the operating system and command line you used to run the planner(s).

3.1.2 Extended Domain

Now consider an elaboration of the domain described in Section 3.1.1: Now some of the room connections may be unavailable ("locked") initially, and become available once a specific artifact ("key") has been collected. As in real life, not every key fits any door, but which key fits which room connection will have to be specified, and the agent will have to collect the key that fits the connection to unlock.

Represent this domain using PDDL, and provide 5 problem instances with 5 to 10 rooms and 3 to 5 artifacts. One of these problem instances should not admit any plan.

Solve all problem instances using the same PDDL planner as for the basic domain.

3.1.3 Essay

Describe using 800-1000 words how you produced the domains and instances, referring also to existing PDDL domains and instances (if any) that you have based your code on, to any tools you have used in producing them (editor, IDE). Also describe all fluents and operators in your domain. Also describe your experience with solving the planning problems. (For example, were the planners adequate and easy to use? Were there any performance or correctness issues?) Finally, reflect on what you achieved on this part of the assignment.

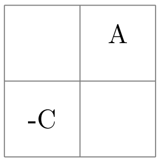

Figure 1: Small example, lower left room is known to be not cold, upper right room is known to be redioactive.

(For instance, did everything work as you expected it? What problems did you encounter? What did you learn?)

3.2 Part 2: Logic Programming

Consider a game consisting of a play area that consists of a number of rooms, some of which are connected to each other; if two rooms are connected, we also call them neighbouring, adjacent, or being next to each other. Rooms can contain nuclear reactors, which cause radioactivity to be measurable in the containing room and all adjacent rooms. Rooms can also contain liquid nitrogen traps, which cause the containing room and all adjacent rooms to be cold. It is possible for a room to contain both a reactor and a trap.

Now consider an agent that has some knowledge about the game state. In particular, the agent knows the exact room layout, the number of reactors, and the number of traps. Furthermore, for each room the agent may have some information on whether it contains a reactor or not, whether it contains a trap or not, whether it is cold or not, and whether it is radioactive or not. This knowledge could stem from having explored some rooms already or it could have been communicated, but the provenance of the knowledge is not important for the assignment. However, the information (apart from the room layout) is *incomplete*, that is, the agent might not know whether a given room is cold or not, radioactive or not, contain a reactor or not, or contain a trap or not.

The task is to compute all possible complete game states, given the partial knowledge of the agent. As an example, look at the situation in Figure 1, where rooms are represented as squares and are connected horizontally and vertically. -C stands for the fact that the agent knows that this room is not cold, whereas the A stands for the fact that tis room is radioactive. Also assume that the agent knows that there is one reactor and one trap. It does not know anything else. So for example for an empty room it is not known whether this room is cold or not, whether it is radioactive or not, whether it has a trap or not, or whether it has a reactor or not.

However the agent can draw some conclusions, for instance that the upper right room must contain the trap: if it was in any other room, this would conflict with the lower left room being not cold. In a similar way, the agent can conclude that the lower left room cannot contain the reactor: as it knows that there is

C -T A R	C T A -R
-C -T A -R	C -T -A -R

C -T A -R	C T A R
-C -T -A -R	C -T A -R

C -T -A -R	C T A -R
-C -T A -R	C -T A R

Figure 2: All solutions for the problem in Figure 1, when it is known that there is 1 reactor and 1 trap.

R	T

	T R

	T
	R

Figure 3: Same as Figure 2, but showing only reactor and trap locations.

only one reactor, placing it in that room would contradict the knowledge that the upper right room is radioactive. However, the reactor could be located in any other room, and this gives rise to the three solutions depicted in Figure 2, and Figure 3 shows the same solutions, but only the trap and reactor locations.

3.2.1 Basic Game

For the basic version, we assume that there is a set of rooms arranged in a rectangular grid (of any size), as in the example seen earlier, that is, each room is adjacent to those rooms that are horizontally or vertically next to it in the grid. In the basic version, no room may contain multiple reactors, and no room may contain multiple traps.

	-C -R -T		
-T		C -R -T	
-R	A -R -T	C -R -T	

Figure 4: Example situation; additional knowledge is that there are two traps and one reactor.

```
-reactor(r13).
-trap(r12).
-cold(r21).          -reactor(r21).   -trap(r21).
stench(r23).         -reactor(r23).   -trap(r23).
cold(r32).           -reactor(r32).   -trap(r32).
cold(r33).           -reactor(r33).   -trap(r33).
totalreactors(1).    totaltraps(2).
adj(r11,r12).        adj(r12,r13).    adj(r13,r14).
adj(r21,r22).        adj(r22,r23).    adj(r23,r24).
adj(r31,r32).        adj(r32,r33).    adj(r33,r34).
adj(r41,r42).        adj(r42,r43).    adj(r43,r44).
adj(r11,r21).        adj(r21,r31).    adj(r31,r41).
adj(r12,r22).        adj(r22,r32).    adj(r32,r42).
adj(r13,r23).        adj(r23,r33).    adj(r33,r43).
adj(r14,r24).        adj(r24,r34).    adj(r34,r44).
```

Figure 5: Input encoding for the problem in Figure 4

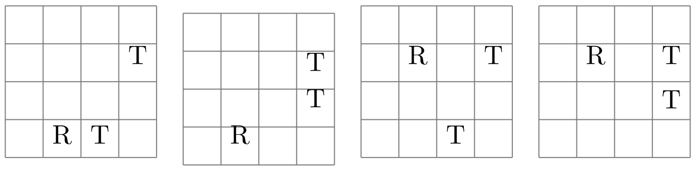

Figure 6: Solution for the example in Figure 4.

Represent this setting using Answer Set Programming (ASP). Assume that the game state is given by facts of predicates `adj/2`, `cold/1`, `radioactive/1`, `trap/1`, `reactor/1`, `totaltraps/1`, and `totalreactors/1`. There will be one fact of `adj/2` for each pair of connected rooms (implying that you cannot rely on the input `adj/2` to be symmetric, so for any fact `adj(x,y).` there will also be a fact `adj(y,x).` in the input). There will be one fact of `cold/1` for each room the agent knows to be cold, while there will be a strongly negated fact of `cold/1` (e.g. `-cold(r).`) for each room the agent knows to be not cold; similar for `radioactive/1`, and also for `reactor/1` and `trap/1`. Finally, there will be exactly one fact of predicate `totalreactors/1`, its argument being the total number of reactors, and `totaltraps/1`, its argument being the total number of traps. The game state of the problem in Figure 4 would be encoded as shown in Figure 5. Note that the choice for labelling the rooms ($r < column >< row >$) is unimportant, you may choose a different way of naming the rooms. The example in Figure 4 admits four solutions, shown in in Figure 6.

Your first task is to specify three game states using this encoding convention in ASP: one should not admit any solution, one should admit exactly one solu-

{reactor(r24),trap(r42),trap(r34), . . . }
{reactor(r24),trap(r42),trap(r43), . . . }
{reactor(r22),trap(r42),trap(r34), . . . }
{reactor(r22),trap(r42),trap(r34), . . . }

Figure 7: Answer sets for the problem in Figure 4

tion, and one should admit more than one solution (and be different from the one in Figure 4, there is also no need to have a 4x4 grid, yours can be smaller or larger).

Your next task is to write an ASP encoding that admits an answer set for each possible solution. The answer sets of your encoding together with an encoding of a game state should correspond one-to-one to the possible configurations. For the example in Figure 4 there should be four answer sets, as shown in Figure 7. The answer sets will contain many more atoms (indicatted by the three dots (. . .) in the figure).

3.2.2 3D Layout

Now consider a 3D version, where rooms are arranged in a 3-dimensional box-like grid. What changes do you need to apply to the encoding conventions of game states of Section 3.2.1? What changes do you need to apply to the domain encoding style of Section 3.2.1?

Again, specify three game states using the encoding convention you came up with: one should not admit any solution, one should admit exactly one solution, and one should admit more than one solution, and then specify an ASP domain encoding that admits exactly one answer set for each solution.

3.2.3 Multiple Reactors per Room

Finally, consider a version of the basic game in of Section 3.2.1 with a rectangular grid, but where each room can hold an arbitrary number of reactors (rather than at most one). There should still not be multiple traps in a room.

What changes do you need to apply to the encoding conventions of game states of Section 3.2.1? What changes do you need to apply to the domain encoding style of Section 3.2.1?

Again, specify three game states using the encoding convention you came up with: one should not admit any solution, one should admit exactly one solution, and one should admit more than one solution, and then specify an ASP domain encoding that admits exactly one answer set for each solution.

3.2.4 Essay

Describe using 800-1000 words how you produced the encodings, referring also to existing ASP code (if any) that you have based your code on, to any tools you have used in producing them (editor, IDE). Also describe all rules and constraints in your code. Also describe your experience with the ASP solvers. (For example, were the solvers adequate and easy to use? Were there any performance or correctness issues?) Finally, reflect on what you achieved in this part of the assignment. (For instance, did everything work as you expected it? What problems did you encounter? What did you learn?)

4 Assessment Criteria

Each of the two parts of this assignment contribute by 50% to the total assignment mark.

Part 1 will be marked along the following criteria:

- Correctness and style of the planning problems for the basic domain 15%
- Correctness and style of the planning domain for the basic domain 20%
- Correctness and style of the planning problems for the extended domain 15%
- Correctness and style of the planning domain for the extended domain 20%
- Essay content 20%
- Essay style 10%

Part 2 will be marked along the following criteria:

- Correctness and style of the ASP instance encodings for the basic game 10%
- Correctness and style of the ASP domain encoding for the basic game 20%
- Correctness and style of the ASP instance encodings for the 3D game 10%
- Correctness and style of the ASP domain encoding for the 3D game 10%
- Correctness and style of the ASP instance encodings for the multi-reactor game 10%
- Correctness and style of the ASP domain encoding for the multi-reactor game 10%
- Essay content 20%
- Essay style 10%

5 Academic Integrity

You are reminded of academic integrity. Any breach of academic integrity (such as plagiarism) can entail sanctions such as the failure of the entire module.

6 Addressed Learning and Ability Outcomes

This assignment assesses learning outcome 1.1, 1.2, 1.3, 1.4 and ability outcomes 2.1 and 2.2 of the module specification.

1. Knowledge and Understanding Outcomes
   - 1.1 explain concepts underlying symbolic AI such as knowledge representation and automated reasoning, and fundamental AI processes such as heuristic search methods
   - 1.2 describe some of central techniques in specific AI areas eg in automated planning and/or machine learning
   - 1.3 understand the role of specific AI research tools and programming languages in AI modelling and experimentation
   - 1.4 discuss some of the major application areas of artificial intelligence; such as in Computer Games or Robot Control
2. Ability Outcomes
   - 2.1 Construct and reason with knowledge representations within a range of AI formalisms, such as rules, action schema, classical logics, Baysian nets etc
   - 2.2 Configure, apply and critically evaluate AI methods, and appropriate tools and techniques, for implementing intelligent systems in application areas such as computer games or robot control