Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

[DΣIM]

PLANNING AND APPROXIMATE REASONING

Planner Exercise 1

Index

Cristòfol Daudén Esmel

MESIIA

14/10/2019

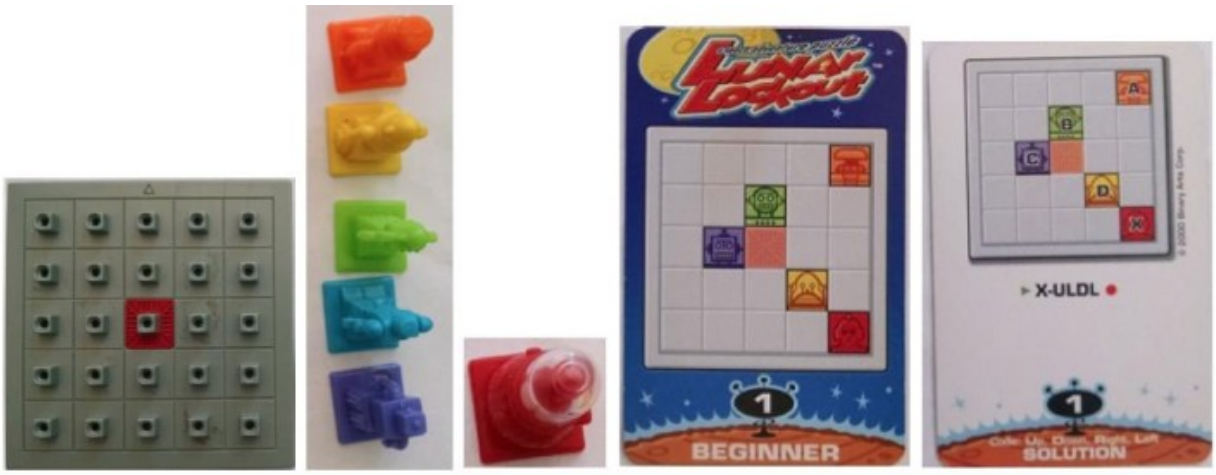
Lunar Lockout Game

The exercise consists of, on the one hand, representing the Lunar Lockout Game world, and in the other hand, the design and implementation of the PDDL files required for the FF planner to solve this problem.

The game

The lunar lockout game has several pieces:

- 5x5 game board with a red square marked in the middle
- 5 helper spacecraft in various colours
- 1 red spacecraft
- cards that specify initial setup position for some subset of spacecraft. On the back of each card is a solution



(a) 5x5 Board (b) Helpers (c) Red (d) Puzzle 1 (e) Solution 1

The goal of this game it to move the red spacecraft to the centre red square. One can move any spacecraft but they are limited to moving up-down or left-right. Whenever a spacecraft moves, it continues moving until it hits another spacecraft.

Task 1: Lunar Lockout world representation

Define two very different ways to represent the lunar lockout world (objects, states, actions, domain axioms). Ensure there is a big difference in the set of possible states between both representations.

Representation 1

The first world objects are:

- The possible X and Y coordinates of the Board (C1, C2, …, C5).
- Five spacecrafts (red, blue, yellow, orange and green).

States:

- **biggerThan(a b)**: means the value of coordinate a is bigger than b's (e.g. C5 > C3).
- **nextto(a b):** means that the coordinate a is next to the coordinate b (e.g. C1-C2)
- **at(a x y):** means spacecraft a is on coordinates (x, y).

Actions:

- **move-up(obj1 obj2 x y y2 y_dest)**: moves the spacecraft "obj1" from (x,y) to (x,y_dest).
  - The move-up action occurs when the spacecraft "obj1" moves up through the board until it hits another spacecraft, "obj2".
  - **Form**: move-up(obj1 obj2 x y y2 y_dest)
  - **Pre**: $at(obj1, x, y) \wedge biggerThan(y, y_{dest})$

    $at(obj2, x, y2) \wedge \neg at(obj2, x, y)$
  - **Add**: $at(obj1, x, y_{dest})$
  - **Del**: $at(obj1, x, y)$
  - **Constraints**: $nextto(y2, y_{dest})$

    $\forall (obj_{aux}, y_{aux}): \neg(at(obj_{aux}, x, y_{aux}) \wedge biggerThan(y, y_{aux}) \wedge biggerThan(y_{aux}, y2))$

The preconditions are that the spacecraft you want to move has to be at the (x,y) position, the spacecraft you are going to hit with, has to be at the same column but in a different row and, because you are moving up through the board (the 0,0 coordinate corresponds to the top,left of the board), the row destiny coordinate (y_dest) has to be lower than the row origin coordinate (y).

The constraints are that the destination coordinate must be one position under the second spacecraft, and I must not be any other spacecraft between the initial position and the spacecraft 2 position.

The result of the action consists of removing spacecraft from position (x,y) and adding it to the (x, y_dest) position.

The process is the same for the rest of actions but modifying the relations between positions (in the down action, value of the destination y coordinate has to be higher than the actual position) and the working coordinate (in the move-up and move-down actions I am modifying the y spacecraft coordinate while in the move-left and move-right actions I am working with the x coordinate).

- **move-down(obj1 obj2 x y y2 y_dest)**: moves the spacecraft "obj1" from (x,y) to (x,y_dest).
  - The move-down action occurs when the spacecraft "obj1" moves down through the board until it hits another spacecraft, "obj2".
  - **Form**: move-up(obj1 obj2 x y y2 y_dest)
  - **Pre**: $at(obj1, x, y) \wedge biggerThan(y_{dest}, y)$ and $at(obj2, x, y2) \wedge \neg at(obj2, x, y)$
  - **Add**: $at(obj1, x, y_{dest})$
  - **Del**: $at(obj1, x, y)$
  - **Constraints**: $nextto(y_{dest}, y2)$

    $\forall (obj_{aux}, y_{aux}): \neg(at(obj_{aux}, x, y_{aux}) \wedge biggerThan(y_{aux}, y) \wedge biggerThan(y2, y_{aux}))$
- **move-right(obj1 obj2 x y x2 x_dest)**: moves the spacecraft "obj1" from (x,y) to (x_dest,y).
  - The move-right action occurs when the spacecraft "obj1" moves right through the board until it hits another spacecraft, "obj2".
  - **Form**: move-right(obj1 obj2 x y x2 x_dest)
  - **Pre**: $at(obj1, x, y) \wedge biggerThan(x_{dest}, x)$ and $at(obj2, x2, y) \wedge \neg at(obj2, x, y)$
  - **Add**: $at(obj1, x_{dest}, y)$
  - **Del**: $at(obj1, x, y)$
  - **Constraints**: $nextto(x_{dest}, x2)$

    $\forall (obj_{aux}, x_{aux}): \neg(at(obj_{aux}, x_{aux}, y) \wedge biggerThan(x_{aux}, x) \wedge biggerThan(x2, x_{aux}))$
- **move-left(obj1 obj2 x y x2 x_dest)**: moves the spacecraft "obj1" from (x,y) to (x_dest,y).
  - The move-left action occurs when the spacecraft "obj1" moves left through the board until it hits another spacecraft, "obj2".
  - **Form**: move-right(obj1 obj2 x y x2 x_dest)
  - **Pre**: $at(obj1, x, y) \wedge biggerThan(x, x_{dest})$ and $at(obj2, x2, y) \wedge \neg at(obj2, x, y)$
  - **Add**: $at(obj1, x_{dest}, y)$
  - **Del**: $at(obj1, x, y)$
  - **Constraints**: $nextto(x2, x_{dest})$

    $\forall (obj_{aux}, x_{aux}): \neg(at(obj_{aux}, x_{aux}, y) \wedge biggerThan(x, x_{aux}) \wedge biggerThan(x_{aux}, x2))$

Representation 2

As the game is quite simple, we only have a 5x5 board and 5 different objects (spacecrafts), the second world representation is quite similar to the first one.

The second world objects, as in the first world, are:

- The possible X and Y coordinates of the Board (C1, C2, …, C5).
- Five spacecrafts (red, blue, yellow, orange and green).

States:

- **biggerThan(a b)**: means the value of coordinate a is bigger than b's (e.g. C5 > C3), it is going to be used for the down and right movements.
- **nextto(a b):** means that the coordinate a is next to the coordinate b (e.g. C1-C2).
- **at(a x y):** means spacecraft a is on coordinates (x, y).
- **between(a b c):** means that position b is between positions a and c (e.g. C1 C3 C5).

Actions:

- **change_row(obj1 obj2 x y y2 y_dest)**: moves the spacecraft "obj1" from (x,y) to (x,y_dest).
  - The change row action occurs when the spacecraft "obj1" moves up or down through the board until it hits another spacecraft, "obj2".
  - **Form**: change_row(obj1 obj2 x y y2 y_dest)
  - **Pre**: $at(obj1, x, y) \wedge at(obj2, x, y2) \wedge \neg at(obj2, x, y)$
  - **Add**: $at(obj1, x, y_{dest})$
  - **Del**: $at(obj1, x, y)$
  - **Constraints**: $\forall (obj_{aux}, y_{aux}): \neg(at(obj_{aux}, x, y_{aux}) \wedge between(y2, y_{aux}, y))$ $(biggerThan(y, y_{dest}) \wedge nextto(y2, y_{dest})) \vee (biggerThan(y_{dest}, y) \wedge nextto(y_{dest}, y2))$

The preconditions are that the spacecraft you want to move has to be at the (x,y) position, the spacecraft you are going to hit with, has to be at the same column but in a different row.

The constraints are that the destination coordinate must be one position under or over the second spacecraft (depending on if the spacecraft is moving up or down), and I must not be any other spacecraft between the initial position and the spacecraft 2 position.

The result of the action consists of removing spacecraft from position (x,y) and adding it to the (x, y_dest) position.

The process is the same for the change column actions but working with the "x" coordinate instead of the "y" coordinate.

- **change_column(obj1 obj2 x y x2 x_dest)**: moves the spacecraft “obj1” from (x,y) to (x_dest,y).
  - The change column action occurs when the spacecraft “obj1” moves left or right through the board until it hits another spacecraft, “obj2”.
  - **Form**: change_column(obj1 obj2 x y x2 x_dest)
  - **Pre**: $at(obj1, x, y) \wedge at(obj2, x2, y) \wedge \neg at(obj2, x, y)$
  - **Add**: $at(obj1, x_{dest}, y)$
  - **Del**: $at(obj1, x, y)$
  - Constraints: $\forall (obj_{aux}, x_{aux}): \neg(at(obj_{aux}, x_{aux}, y) \wedge between(x2, x_{aux}, x))$ $(biggerThan(x, x_{dest}) \wedge nextto(x2, x_{dest})) \vee (biggerThan(x_{dest}, x) \wedge nextto(x_{dest}, x2))$

Task 2: Possible States

Explain the number of possible states for both representations that you came up with. Do they also differ in the number of feasible states? Explain why each representation would be preferable to the other.

Both representations have the same number of possible states as they are determined by the relative positions of the different spacecrafts on the board. As we are working on a 5x5 board, there are 25 feasible positions for each one of the spacecrafts with the restriction that it can not be more than one spacecraft AT the same position. Taking this into account, when the first spacecraft has been set to a certain position, there only remain 24 different positions for the second spacecraft, 23 for the third, 22 for the forth and 21 for the last one, leading to a total states number of 25x24x23x22x21 = 6375600.

Task 3: Writing the PDDL files

Write the PDDL files required for a planner of your choice to solve this problem (Hint: a state-space search is likely the most simple). We should be able to run your code with something like the following command: /lunar-lockout <domain.pddl> <problem.pddl>. The <domain .pddl> should define the lunar lockout world actions and predicates. If you wish, you can hardcode the domain representation into your suggested planner but be sure to state this in your README. The <problem .pddl> contains the objects, init state, and goal state. The output of your planner should be a sequence of actions that solves the given puzzles. Explain how we should interpret the output. It is OK to use a more compact representation.

To run the two different implemented domains in order to solve the presented problems with the FF Planer you have to type the following command in the terminal:

FF-X/ff -o [domain_file] -f [problem_file]

The results achieved by using the first implemented domain (*lunarLockoutGame.pddl*) should be interpreted as follows:

action number : name of the action
spacecraft that is going to be moved (s1)
spacecraft with which you are going to crash (s2)
s1 row
s1 column
s2 column or row depending on the kind of action (row if we are doing a up-down movement and column if we are doing a left-right movement)
destination row or column, depending of the kind of move

For a better comprehension you can only take into account the first 3 arguments (action, spacecrat1 and spacecrat2), which say that spacecraft1 will move in the direction specified by the action until it crashes with spacecrat2.

The result using the first domain with the first problem (*lunarLockoutGame_problem.pddl*) looks as follows:

0: MOVE-UP RED ORANGE C5 C5 C1 C2
1: MOVE-LEFT RED GREEN C5 C2 C3 C4
2: MOVE-DOWN RED YELLOW C4 C2 C4 C3
3: MOVE-LEFT RED BLUE C4 C3 C2 C3
Time: 0,14s

On the other hand, the results achieved by using the second implemented domain (*lunarLockoutGame_V2.pddl*) could be harder to interpreted as it only works with two different actions. In order to better understand the results you can also only take into account the first 3 arguments (action, spacecrat1 and spacecrat2), which say that spacecraft1 will move in the direction specified by the action (moving through its actual row or column) until it crashes with spacecrat2. If you need more information because you can not visualize the board, to know if the spacecraft is moving up or down, for example, you only have to compare the *s1 row* with the *destination row.* If

the destination row value is higher than the actual row value, you are moving down, if not, you are moving up.

The result using the second domain with the first problem (*lunarLockoutGame_problem_V2.pddl*) looks as follows:

0: CHANGE_ROW RED ORANGE C5 C5 C1 C2
1: CAHNGE_COLUMN RED GREEN C5 C2 C3 C4
2: CHANGE_ROW RED YELLOW C4 C2 C4 C3
3: CAHNGE_COLUMN RED BLUE C4 C3 C2 C3
Time: 0,09s

More Results

The result using the first domain with the second problem (*lunarLockoutGame_problem2.pddl*) looks as follows:

0: MOVE-DOWN BLUE ORANGE C2 C2 C5 C4
1: MOVE-RIGHT BLUE GREEN C2 C4 C4 C3
2: MOVE-RIGHT RED YELLOW C1 C1 C4 C3
3: MOVE-DOWN RED BLUE C3 C1 C4 C3
Time: 0,15s

The result using the second domain with the second problem (*lunarLockoutGame_problem2_V2.pddl*) looks as follows:

0: CHANGE_ROW BLUE ORANGE C2 C2 C5 C4
1: CAHNGE_COLUMN BLUE GREEN C2 C4 C4 C3
2: CAHNGE_COLUMN RED YELLOW C1 C1 C4 C3
3: CHANGE_ROW RED BLUE C3 C1 C4 C3
Time: 0,09s

There are also many benchmark problem files to check that the implemented domains take into account specific situations such as when is another spacecraft in the middle of the path or when another spacecraft is on the goal position. See:

Benchmark 1

benchmark1.pddl and ***benchmark1_V2.pddl*** files

Looks as:

		Y		
B		G		
		R		

The solution using domain 1 is:

0: MOVE-LEFT GREEN BLUE C3 C4 C1 C2

1: MOVE-UP RED YELLOW C3 C5 C2 C3

Using domain 2 is:

0: CAHNGE_COLUMN GREEN BLUE C3 C4 C1 C2

1: CHANGE_ROW RED YELLOW C3 C5 C2 C3

Benchmark 2

benchmark2.pddl and ***benchmark2_V2.pddl*** files

Looks as:

		Y		
B		G		
		R		

The solution using domain 1 is:

0: MOVE-LEFT GREEN BLUE C3 C3 C1 C2

1: MOVE-UP RED YELLOW C3 C5 C2 C3

Using domain 2 is:

0: CAHNGE_COLUMN GREEN BLUE C3 C3 C1 C2

1: CHANGE_ROW RED YELLOW C3 C5 C2 C3