# AUTOMATED PLANNING

***Date: 04/01/2016***

***Developed by:***

***Prerana Shamsundar Punjabi***

# Table of Contents

Executive Summary ..... 3
1 Problem Description ..... 4
2 Conclusion ..... 8
3 Appendix ..... 9

# Executive Summary

Automatic Planning is one of the fundamental sub-areas of Artificial Intelligence, concerned with algorithms that can generate strategies of action for arbitrary autonomous agents in arbitrary environments. In classical planning, the actions and environment are assumed to be deterministic.

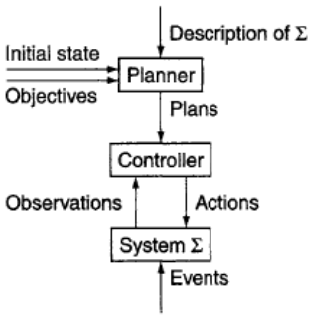
Planning is the reasoning side of acting. A conceptual model is a simple theoretical device for describing the main elements of a problem. The model used is called state-transition systems (also called discrete-event systems).

Formally, a restrictive state-transition system is a triple E = (S, A, γ), where:

- S = {Sl, S2 . . . . } is a finite or recursively enumerable set of states;
- A = {al, a2,...} is a finite or recursively enumerable set of actions;

Actions are transitions that are controlled by the plan executor. If a is an action and y(s, a) is not empty, then action a is applicable to state s; applying it to s will take the system to some state in y (s, a). A planning problem for a restricted state-transition system E--(S, A, F) is defined as a triple P = (E, so, g), where so is an initial state and g corresponds to a set of goal states.

Classical planning refers generically to planning for restricted state-transition systems.



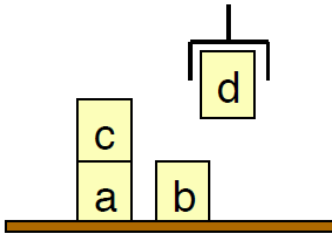

In Classical Representation: States are represented as sets of logical atoms that are true or false within some interpretation. Actions are represented by planning operators that change the truth values of these atoms.

In this lab work we have used FF – Fast Forward tool (domain independent planning system) tool to find efficient solutions.

# 1 Problem Description

**Problem Description 1:**

The Aim of this problem was to stack 4 blocks one on top of the other. Such that block d is first placed on the table and block c was then placed over it and then block b and at last block a was placed on the top of the stack.




The problem consists of 4 blocks and 1 robot.

TABLE 1: PREDICATES USED

| predicates | description |
|---|---|
| clear | the block is accessible |
| on-table | the block is on the table |
| arm-empty | the arm is empty |
| On | the block ?x is over ?y |
| holding | the arm is holding the block |

TABLE2: ACTION AND PRECONDITIONS USED

| actions | description |
|---|---|
| pickup | picks up a block that is accessible – the preconditions are set accordingly<br>1. The block is accessible<br>2. Robot arm is empty<br>3. The block is on table |
| putdown | putdown a block on the table – the preconditions are set accordingly<br>1. The robot is holding the block |
| stack | a block placed by the arm on an accessible block – the preconditions are set accordingly<br>1. The robot is holding a block<br>2. The other block - upon which the held block must be placed is accessible |
| unstack | pick up the block that is currently placed on another block – the preconditions are set accordingly<br>1. The block to be picked up is accessible<br>2. The block is currently placed on another block<br>3. The robot arm is free |

This is a simple problem where in first we define how the initial layout looks / initial states.

- Block C and Block D are accessible
- Block C is on Block A
- Block D is on Block B
- Block A and Block B are on-table

**Solution**:

- First, Block D is placed onto the Table, Block B becomes accessible
- Block C is picked up and A becomes accessible
- Block C is placed onto Block D
- Block B is placed onto Block C
- Lastly Block A is placed on Block B
- Arm is empty

**Problem Description 2:**

A robot is used for moving objects from some rooms to another room. The robot should collects all the objects encountered while traversing from the rooms and unload all objects in the final room F. The characteristics of the robot are the following:

- The robot can go from a room to another according to the plan
- The robot can pick an object and can keep it in a container attached to its body. **Remark**: for sake of simplicity, consider that once the robot picked the object it will be in the container.

The problem statement defined 3 sections

1. The robot can move from one room to the other
2. The robot can pick up the objects it encountered while traversing and place it in its container (capacity is infinite)
3. The capacity of the container is taken into account

TABLE 3: PREDICATES USED

| Predicates | Description |
|---|---|
| in-room | The robot is in the room |
| Adjacent | Room A is adjacent to room B |
| has-obj | The room has an object |
| in container | The object is in the container |

TABLE4: ACTION AND PRECONDITIONS USED

| Actions | Description |
|---|---|
| move_room1-room2 | move from room 1 to room 2 – the preconditions are set accordingly<br>1. The robot is in room 1<br>2. Room 2 is adjacent to room 1 |

| | |
|---|---|
| Pickup | Pickup the object in the room – the preconditions are set accordingly<br>1. The robot is in that room<br>2. The room has object<br>3. The capacity is less than the maximum capacity |
| Putdown | Unload the objects from the container – the preconditions are set accordingly<br>1. The robot is in room F<br>2. The container has objects |

In this problem we define how the initial layout looks / initial states are:

- Robot is in Room A
- Room A is adjacent to room B
- Room B is adjacent to room F
- Room A is adjacent to room C
- Room B is adjacent to room D
- Room C is adjacent to room E
- Room B is adjacent to room A
- Room F is adjacent to room B
- Room C is adjacent to room A
- Room D is adjacent to room B
- Room E is adjacent to room C
- Room A, B, C, D has 1 object each
- Room E has 2 objects
- Number of initial moves is 0
- Capacity is equal to 3

Goal is to reach **Room F** and all objects should be finally located in room F and we need to minimize the number moves used to traverse to collect these objects.

Each time –

1. the robot moves from one room to another the number of moves is increased by 1
2. The robot picks up an object the capacity is decreased by 1
3. The robot unloads the object the capacity is increased by 1

Following is the **solution obtained** is attached as a text file below

output.txt

ff: search configuration is Enforced Hill-Climbing, then A*epsilon with weight 5.
Metric is ((1.00*[RF1](num-of-moves)) - () + 0.00)
COST MINIMIZATION DONE (WITHOUT cost-minimizing relaxed plans).
Cueing down from goal distance: 17 into depth [1]
16 [1]
15 [1][2]
14 [1][2][3]
13 [1]
12 [1]

```
11          [1]
10          [1][2]
9           [1][2]
8           [1][2][3][4]
7           [1][2]
6           [1]
5           [1]
4           [1]
3           [1]
2           [1]
1           [1]
0
ff: found legal plan as follows
step    0: pickup a o1
        1: pickup a o2
        2: move_room1-room2 a c
        3: move_room1-room2 c d
        4: pickup d o4
        5: move_room1-room2 d b
        6: move_room1-room2 b f
        7: putdown f o1
        8: putdown f o2
        9: putdown f o4
       10: move_room1-room2 f b
       11: pickup b o6
       12: move_room1-room2 b f
       13: putdown f o6
       14: move_room1-room2 f b
       15: move_room1-room2 b a
       16: move_room1-room2 a c
       17: pickup c o5
       18: move_room1-room2 c e
       19: pickup e o3
       20: move_room1-room2 e c
       21: move_room1-room2 c a
       22: move_room1-room2 a b
       23: move_room1-room2 b f
       24: putdown f o3
       25: putdown f o5
plan cost: 14.000000
```

It takes about 25 moves to complete the accomplish the task as we have defined the capacity to carry the ball as 3. Hence the robot will return to the room f when the capacity of the container is met, it will unload and then continue to collect the balls from the room. This is followed until all balls are collected and unloaded. It was also observed that when the robot was given unlimited capacity of the container the task was accomplished in only 19 moves (steps) .

# 2 Conclusion

The session gave us a unique exposure to realize how automated planning can be done for simple problems.

The trick is to think in very simplistic way, clearly mention the objects, initial condition and states the final goal to be achieved. List the task / predicates that are required to accomplish simple task. Lastly mention actions that take place, with defining preconditions for each action and post conditions (after the action has occurred the final state) / effects.

We were able to use metric also that allows us to do complex decisions as optimization of the algorithm used. By defining which parameter we would like to optimize. (Here we choose to optimize the number of movements the robot would have to make to collect all objects).

FF is an easy tool and we could solve the problems efficiently.

# 3 Appendix

1. Code for first excersise: (Stack 4 blocks)

```
;; a simple blockword problem with 1 robot and 4 blocks
(define (problem blkPb2)
 (:domain blocksworld)
 (:objects
  a b c d - block)
 (:init
   (clear c) (clear d)
   (on-table a) (on-table b)
   (on c a)
   (on d b)
   (arm-empty)
 )
;; the task is to put c on d, b on c, a on b and d on the table
 (:goal
   (and   (clear a)
          (on-table d)
              (on c d) (on b c) (on a b)
          (arm-empty)

   )))
```

2. Code for first excersise: (Stack 4 blocks)

----- Robo-operators -----

```
;; Specification in PDDL1 of the blockworld domain
(define (domain blocksworld)

;(:requirements :strips :typing )
(:types room object)
(:predicates
  (in-room ?room1 - room)             ; in part room1 of the room
  (adjacent ?room1 ?room2 - room)      ; room1 is adjacent to
  (has-obj ?room1 - room ?obj - object) ; the room has object
  (in_container ?obj - object)         ; the ball is in container
)

(:functions
          (num-of-moves)
          (capacity)
)
```

```
;; move from one part to other

(:action move_room1-room2
   :parameters (?room1 ?room2 - room)
   :precondition (and (in-room ?room1) (adjacent ?room1 ?room2) )
   :effect (and (not (in-room ?room1)) (in-room ?room2) (increase (num-of-moves) 1) )
)

;; pick up the object
(:action pickup
  :parameters (?room1 - room ?obj - object)
  :precondition (and (has-obj ?room1 ?obj) (in-room ?room1) (> (capacity) 0) )
  :effect (and (not(has-obj?room1 ?obj)) (in_container ?obj) (decrease (capacity) 1) )
)

;; putdown the object
(:action putdown
  :parameters  (?room1 - room ?obj - object)
  :precondition (and   (in_container ?obj)  (in-room ?room1)    )
  :effect (and   (not(in_container ?obj))   (has-obj ?room1 ?obj)  (increase (capacity) 1)   )
)

)
```

----- Robo-Pb1 -----

```
;; a simple blockword problem with 1 robot and 3 blocks
(define (problem roboPb1)
  (:domain blocksworld)
  (:objects
   a b c d e f - room
   o1 o2 o3 o4 o5 o6 - object

)
  (:init

     (in-room a)

     (adjacent a b)
     (adjacent b a)

     (adjacent b f)
     (adjacent f b)

     (adjacent a c)
     (adjacent c a)
```

```
    (adjacent b d)
    (adjacent d b)

    (adjacent c e)
    (adjacent e c)

    (adjacent c d)
    (adjacent d c)

    (has-obj a o1)
    (has-obj a o2)
    (has-obj b o6)
    (has-obj c o5)
    (has-obj d o4)
    (has-obj e o3)

    (= (num-of-moves) 0)
    (= (capacity) 3)

  )
;; the task is to put a on b, b on c and c on the table
  (:goal
    (and

        (in-room f)
        (has-obj f o1)
        (has-obj f o2)
        (has-obj f o3)
        (has-obj f o4)
        (has-obj f o5)
        (has-obj f o6)))

        (:metric minimize(num-of-moves))
    )
```