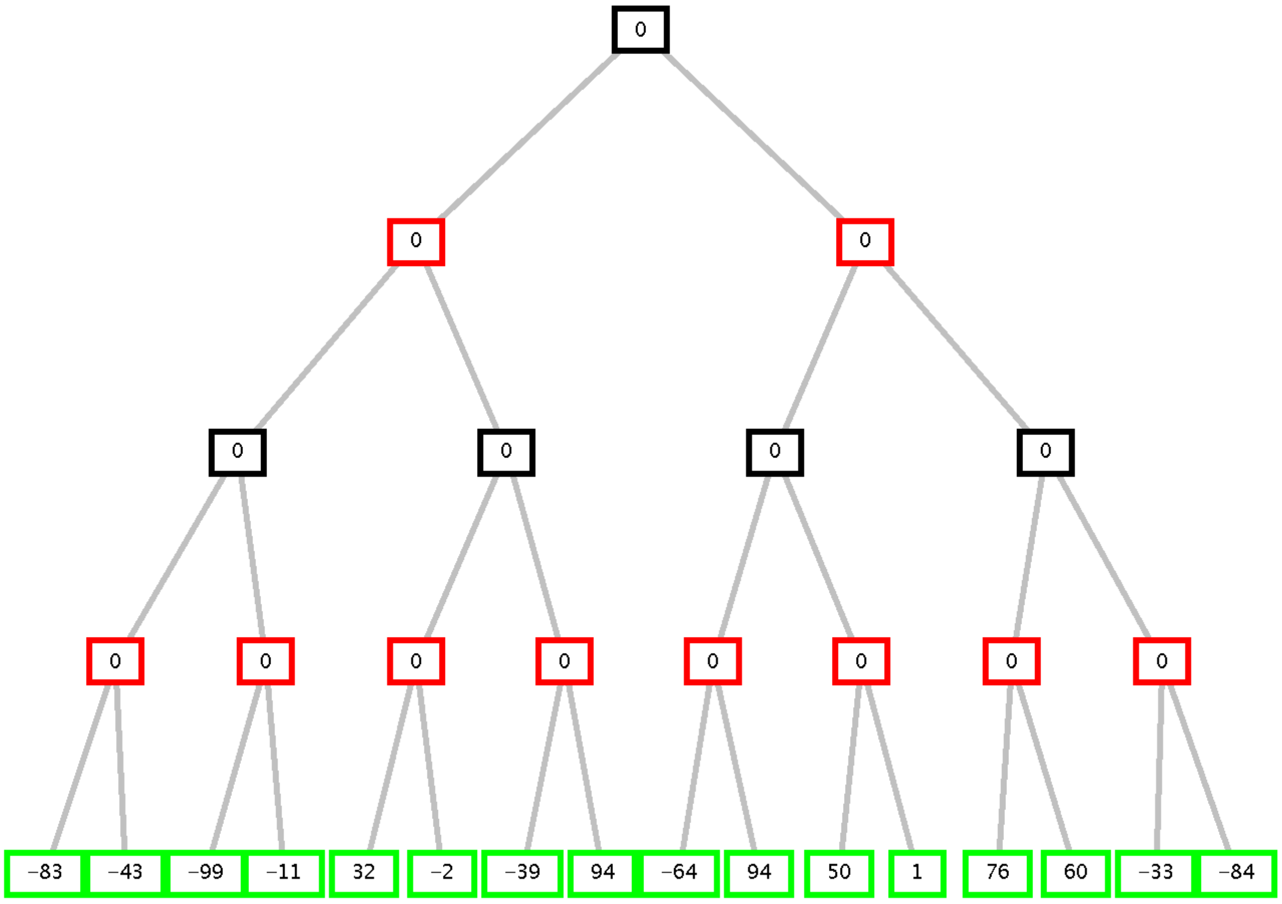
EXAM OF FUNDAMENTALS OF AI – FIRST MODULE
23/01/2020
PROF. MICHELA MILANO

Exercise 1

Consider the following game tree where the first player is *MAX*. Show how the *min-max* algorithm works and show the *alfa-beta* cuts. Also, show which is the proposed move for the first player.



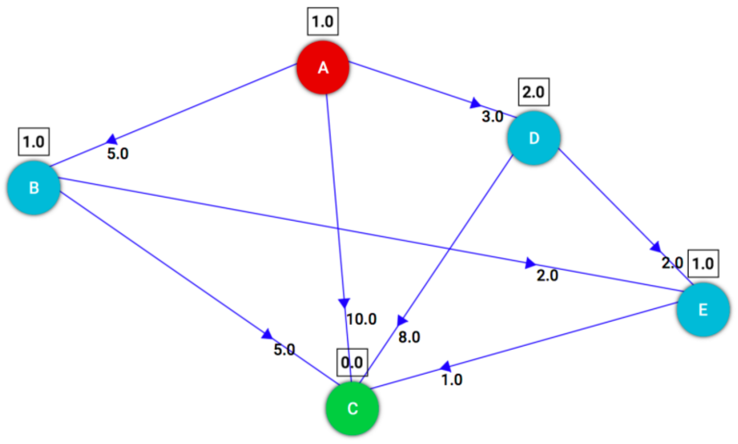
Exercise 2

Four friends, Anna, Bea, Carla and Didi, went shopping and buying scratch cards in a mall. We know that in the entrance the friends each had an amount different from those of the others, and equal to 100, or 200, or 300, or 400 euros. Leaving the mall, we take friends to tell each other:
Anna: I earned! I went out with 100 euros more than what Bea had when she entered;
Didi: I don't have a euro anymore! I spent the same sum that Bea spent; but something remained to her as she was the one with the most money among us!
Carla: I offer you something, I earned a sum equal to twice what Didi spent and now I have the same money as Anna has.
Each of the friends went out with a sum of 0, 100, 200, 300, 400 or 500 euros.

Model the problem as a CSP, where the variables represent the money the friends have before shopping (A, B, C, D, respectively), and the money they have after shopping (A', B', C' and D').
Then apply the arc-consistency technique to the initial network (**consider for this purpose only the unary and binary constraints** in the application of the arc-consistency).

Exercise 3

Consider the following graph, where A is the initial node and C the goal node, and the number associated with the arcs is the cost of the operator to go from the starting node to the arrival node of the arc. Next to each node, in a square, the heuristic estimate of its distance from the goal node is also indicated.



a) Apply the depth-first search, and draw the developed search tree indicating for each node n the cost g(n) and the expansion order; in case of non-determinism, choose the nodes to be expanded according to the alphabetical order.
b) Apply the A* search, and draw the developed search tree indicating for each node n the function f(n) and the expansion order. In the case of non-determinism, choose the nodes to be expanded according to the alphabetical order. Consider as heuristic h(n) the one indicated in the square next to each node in the figure, that is:
h (A) = 1, h (B) = 1, h (D) = 2, h (E) = 1, h (C) = 0. Is the heuristic h defined in this way admissible?
What advantage is obtained by applying A*, compared to the outcome of the depth-first search?

Exercise 4

Consider the problem of moving an object initially on the table to room1 by using a robot.
We have the following actions available:

Loading an object
load (X, Pos)
PREC: at (robot, Pos), at (X, Pos), robotfree
EFFECT: in (robot, X), ¬robotfree

Moving an object
carry (X, Pos1, Pos2)
PREC: at (robot, Pos1), in (robot, X)
EFFECT: at (robot, Pos2), ¬ at (robot, Pos1)

Unloading an object
deliver (X, Pos)
PREC: at (robot, Pos), in (robot, X)
EFFECT: at (X, Pos), ¬ in (robot, X), robotfree

Initial state: **at (robot, table), in (robot, ogg)**
Goal state: **at (ogg, room1)**

Show how the STRIPS algorithm finds a solution (only one successful path in the search tree should be shown).

Exercise 5

1) Model the action **load** (preconditions, effects and frame axioms), the initial state and the goal of the exercise 4 using the Kowalsky formulation

1) Show two levels of graph plan when applied to exercise 4.

1) What is hierarchical planning and explain the method presented during the course.

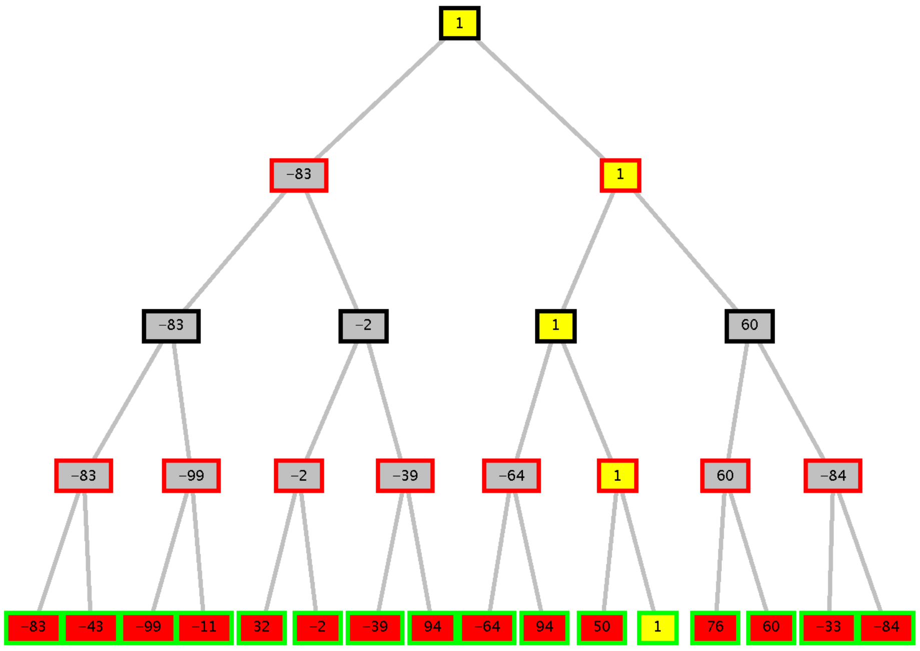
1) What is Particle Swarm Optimization and which are its main features?

2) What are non-informed search strategies? Describe the strategies that have been presented during the course.

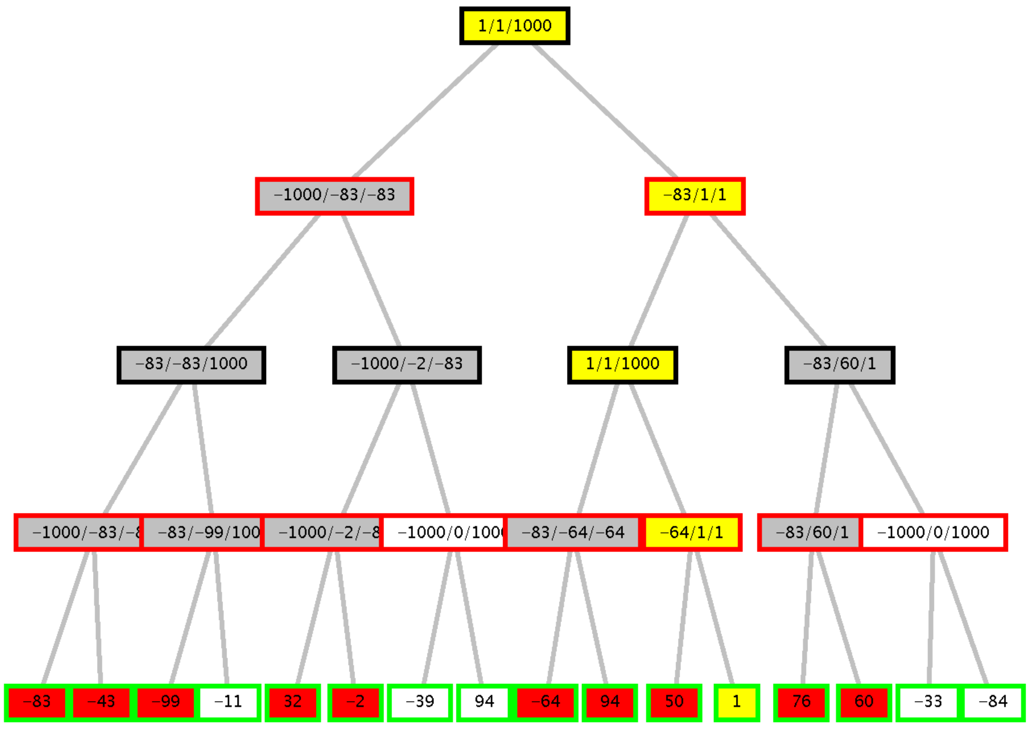
Solution

Exercise 1

Min-max



Alfa-beta:



Exercise 2

Variables: initial amount of money for each friend A, B, C, D in input and final amount of money A’, B’, C’, D’

Domains: (1..4) for A, B, C, D and integer value for A’, B’, C’, D’
A:: [1,2,3,4]
B:: [1,2,3,4]
C:: [1,2,3,4]
D:: [1,2,3,4]
A’:: [0,1,2,3,4,5]
B’:: [0,1,2,3,4,5]
C’:: [0,1,2,3,4,5]
D’:: [0,1,2,3,4,5]

Constraints:
Variables A, B, C, D different:
$A \neq B \neq C \neq D$

Anna (and Bea):
$A' > A$
$A' = 1 + B$

Didi:
$D' = 0$
$D - D' = B - B'$
$D' < D$
$B' < B$
$B > A$,
$B > C$
$B > D$
$B' > 0$

Carla:
$C' > C$
$C' - C = 2^* (D - D') = 2^*D$ *since* $D'=0$
$C' = A'$

Arc-consistency

By applying arc-consistency, the variables domains are reduced as follows:

A:: [1,2,3]
B:: [2,3,4]
C:: [1,2,3]
D:: [1,2,3]
A’:: [2,3,4,5]
B’:: [1,2,3]
C’:: [2,3,4,5]
D’:: [0]

***Applied reasoning*:**

We start with the following domains:

A:: [1,2,3,4]
B:: [1,2,3,4]
C:: [1,2,3,4]
D:: [1,2,3,4]
A’:: [0,1,2,3,4,5]
B’:: [0,1,2,3,4,5]
C’:: [0,1,2,3,4,5]
D’:: [0,1,2,3,4,5]

And with the following set of unary:

1) *D’ = 0*
2) *B’ > 0*

and binary constraints:

a) *A’ > A*
b) *A’ = 1 + B*
c) *D’ < D*
d) *B’ < B*
e) *B > A*
f) *B > C*
g) *B > D*
h) C’ > C
i) *C’ = A’*

Unary constraints need no iteration, as the domain of the variable involved can only be restricted in the following iterations, so we can enforce them before the first iteration.

Apply (1) – **D’ = 0**

- The only values of D’ which admit a solution are [0].

Apply (2) – **B’ > 0**

- The only values of B’ which admit a solution are [1,2,3,4,5].

New domains:
A:: [1,2,3,4]
B:: [1,2,3,4]
C:: [1,2,3,4]
D:: [1,2,3,4]
A’:: [0,1,2,3,4,5]
B’:: [1,2,3,4,5]
C’:: [0,1,2,3,4,5]
D’:: [0]

Now we can proceed with binary constraints until we reach a convergence.

First Iteration:

Apply (a) – **A’ > A**

- For each A’, is there an A? No. The only values of A’ which admit a solution are [2,3,4,5].
- For each A, is there an A’? Yes.

New domains:
A:: [1,2,3,4]
B:: [1,2,3,4]
C:: [1,2,3,4]
D:: [1,2,3,4]

A’:: [2,3,4,5]
B’:: [1,2,3,4,5]
C’:: [0,1,2,3,4,5]
D’:: [0]

Apply (b) – **A’ = 1 + B**

- For each A’, is there a B? Yes.
- For each B, is there an A’? Yes.

Domains not changed.

Apply (c) – **D’ < D**

- For each D’, is there a D? Yes.
- For each D, is there an D’? Yes.

Domains not changed.

Apply (d) – **B’ < B**

- For each B’, is there a B? No. The only values of B’ which admit a solution are [1,2,3].
- For each B, is there a B’? No. The only values of B which admit a solution are [2,3,4].

New domains:
A:: [1,2,3,4]
B:: [2,3,4]
C:: [1,2,3,4]
D:: [1,2,3,4]
A’:: [2,3,4,5]
B’:: [1,2,3]
C’:: [0,1,2,3,4,5]
D’:: [0]

Apply (e) – **B > A**

- For each B, is there an A? Yes.
- For each A, is there a B? No. The only values of A which admit a solution are [1,2,3].

New domains:
A:: [1,2,3]
B:: [2,3,4]
C:: [1,2,3,4]
D:: [1,2,3,4]
A’:: [2,3,4,5]
B’:: [1,2,3]
C’:: [0,1,2,3,4,5]
D’:: [0]

Apply (f) – **B > C**

- For each B, is there a C? Yes.
- For each C, is there a B? No. The only values of C which admit a solution are [1,2,3].

New domains:
A:: [1,2,3]
B:: [2,3,4]
C:: [1,2,3]
D:: [1,2,3,4]
A’:: [2,3,4,5]
B’:: [1,2,3]
C’:: [0,1,2,3,4,5]
D’:: [0]

Apply (g) – **B > D**

- For each B, is there a D? Yes.
- For each D, is there a B? No. The only values of D which admit a solution are [1,2,3].

New domains:

A:: [1,2,3]
B:: [2,3,4]
C:: [1,2,3]
D:: [1,2,3]
A':: [2,3,4,5]
B':: [1,2,3]
C':: [0,1,2,3,4,5]
D':: [0]

Apply (h) – **C' > C**

- For each C', is there a C? No. The only values of C' which admit a solution are [2,3,4,5].
- For each C, is there a C'? Yes.

New domains:

A:: [1,2,3]
B:: [2,3,4]
C:: [1,2,3]
D:: [1,2,3]
A':: [2,3,4,5]
B':: [1,2,3]
C':: [2,3,4,5]
D':: [0]

Apply (i) – **C' = A'**

- For each C', is there an A'? Yes.
- For each A', is there a C'? Yes.

Domains not changed.

Second Iteration (since the domains have changed during the last one):

Apply (a) – **A' > A**

- For each A', is there an A? Yes.
- For each A, is there an A'? Yes.

Domains not changed.

Apply (b) – **A' = 1 + B**

- For each A', is there a B? Yes.
- For each B, is there an A'? Yes.

Domains not changed.

Apply (c) – **D' < D**

- For each D', is there a D? Yes.
- For each D, is there an D'? Yes.

Domains not changed.

Apply (d) – **B' < B**

- For each B', is there a B? Yes.
- For each B, is there a B'? Yes.

Domains not changed.

Apply (e) – **B > A**
- For each B, is there an A? Yes.
- For each A, is there a B? Yes.

Domains not changed.

Apply (f) – **B > C**
- For each B, is there a C? Yes.
- For each C, is there a B? No. The only values of C which admit a solution are [1,2,3].

Domains not changed.

Apply (g) – **B > D**
- For each B, is there a D? Yes.
- For each D, is there a B? Yes.

Domains not changed.

Apply (h) – **C' > C**
- For each C', is there a C? Yes.
- For each C, is there a C'? Yes.

Domains not changed.

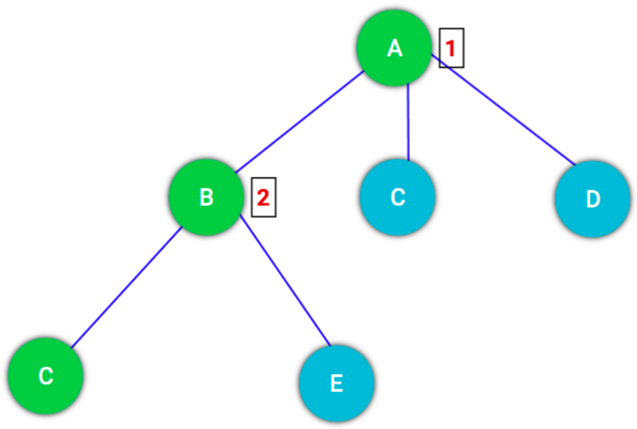
Apply (i) – **C' = A'**
- For each C', is there an A'? Yes.
- For each A', is there a C'? Yes.

Domains not changed.

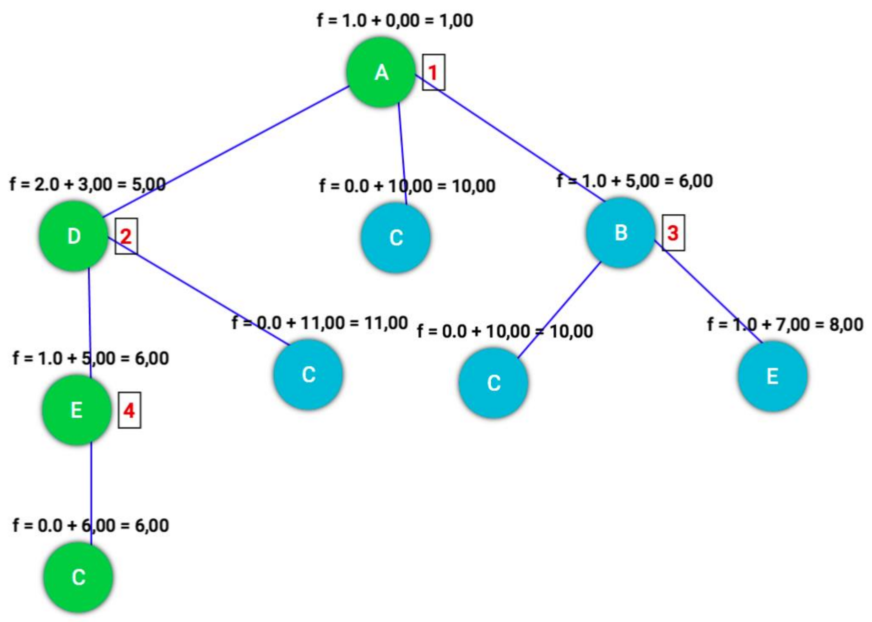
Algorithm termination.

Exercise 3

Depth-First

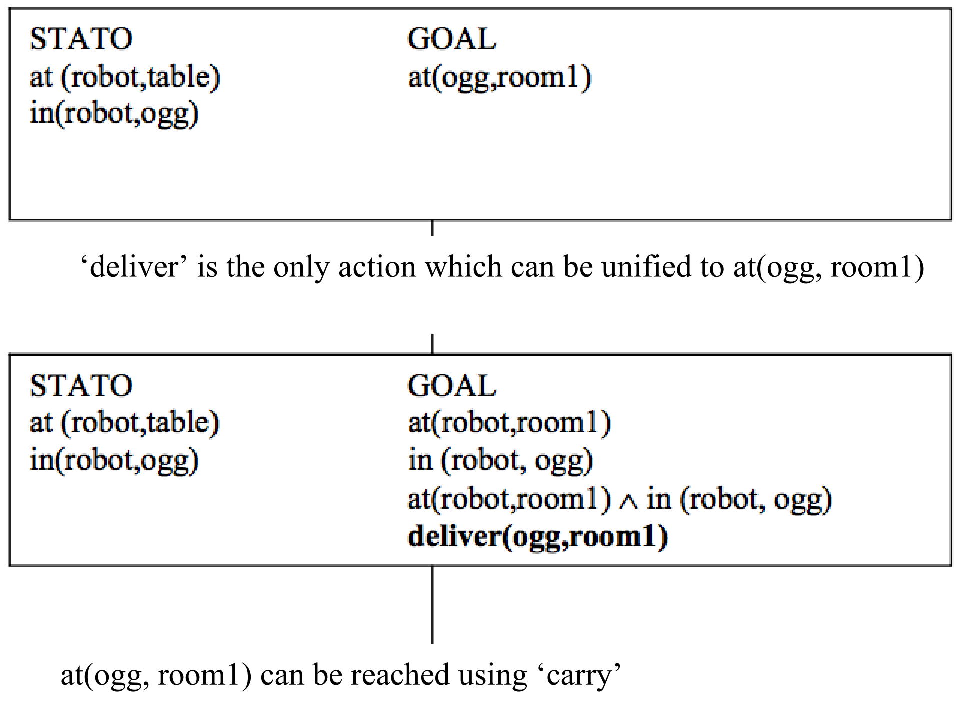


Cost of path found (in green), ABC equal to 10.



With A*, path cost found (in green) - ADEC equal to 6 (optimal path). Admissible heuristics because it never overestimates the cost of reaching the goal, regardless of which node of the graph you consider as starting point.

Exercise 4

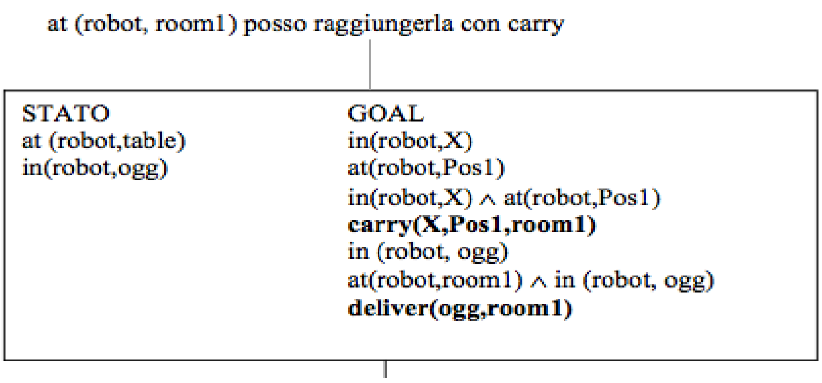


STATO	GOAL
at (robot,table)	at(ogg,room1)
in(robot,ogg)	

'deliver' is the only action which can be unified to at(ogg, room1)

STATO	GOAL
at (robot,table)	at(robot,room1)
in(robot,ogg)	in (robot, ogg)
	at(robot,room1) ∧ in (robot, ogg)
	deliver(ogg,room1)

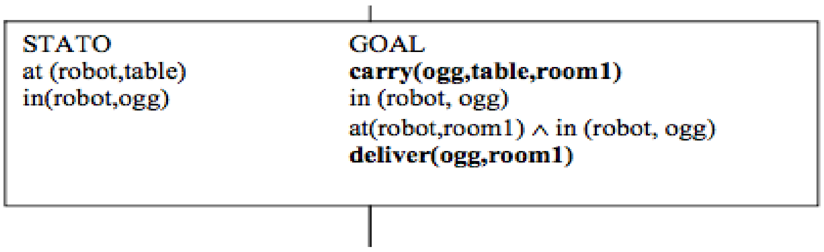
at(ogg, room1) can be reached using 'carry'



at (robot, room1) posso raggiungerla con carry

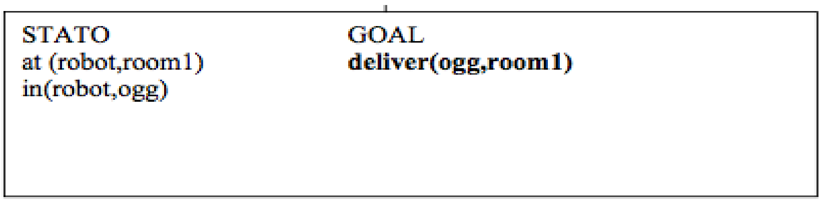
STATO	GOAL
at (robot,table)	in(robot,X)
in(robot,ogg)	at(robot,Pos1)
	in(robot,X) $\wedge$ at(robot,Pos1)
	carry(X,Pos1,room1)
	in (robot, ogg)
	at(robot,room1) $\wedge$ in (robot, ogg)
	deliver(ogg,room1)

Both the preconditions of ‘carry’ are met by grounding X/ogg and Pos1/table

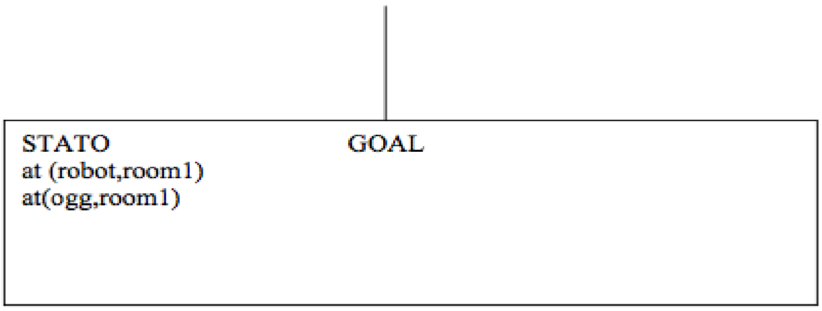


STATO	GOAL
at (robot,table)	**carry(ogg,table,room1)**
in(robot,ogg)	in (robot, ogg)
	at(robot,room1) $\wedge$ in (robot, ogg)
	deliver(ogg,room1)

‘deliver’ preconditions are met in the current state



STATO	GOAL
at (robot,room1)	**deliver(ogg,room1)**
in(robot,ogg)	



STATO	GOAL
at (robot,room1)	
at(ogg,room1)	

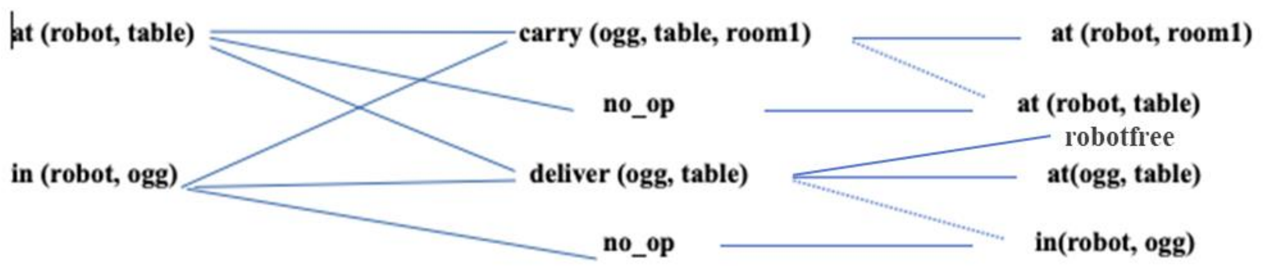
Exercise 5

1)

```
holds(at(robot,table),s0).
holds(in(robot,ogg), s0).

holds(in(robot,X), do(load(X,Pos),S))
pact(load(X,Pos),S):- holds(at(robot,Pos), S), holds(at(X,Pos),S), holds(robotfree,S).
holds(V,do(load(X,Pos),S)):- holds(V,S), V\=robotfree.
```

2)



The following actions are incompatibl: carry(ogg, table, room1)-no_op, deliver(ogg, table)-no_op, carry(ogg, table, room1)- deliver(ogg, table).

There are also inconsistent propositions: at(robot, room1)-at(robot, table), at(ogg, table)-in(robot,ogg), at(robot, room1)-at(ogg, table), robotfree-at(robot, room1), robotfree-in(robot, ogg).