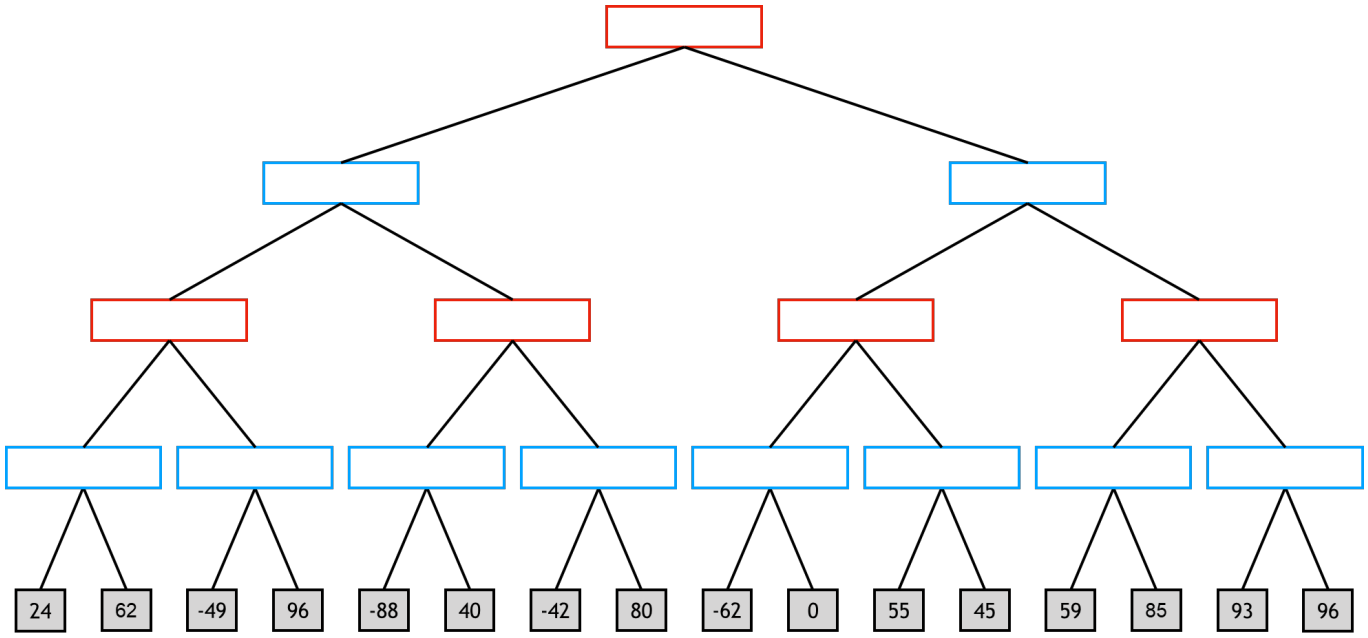
EXAM OF FUNDAMENTALS OF AI – FIRST MODULE
SIMULATION OF EXAM

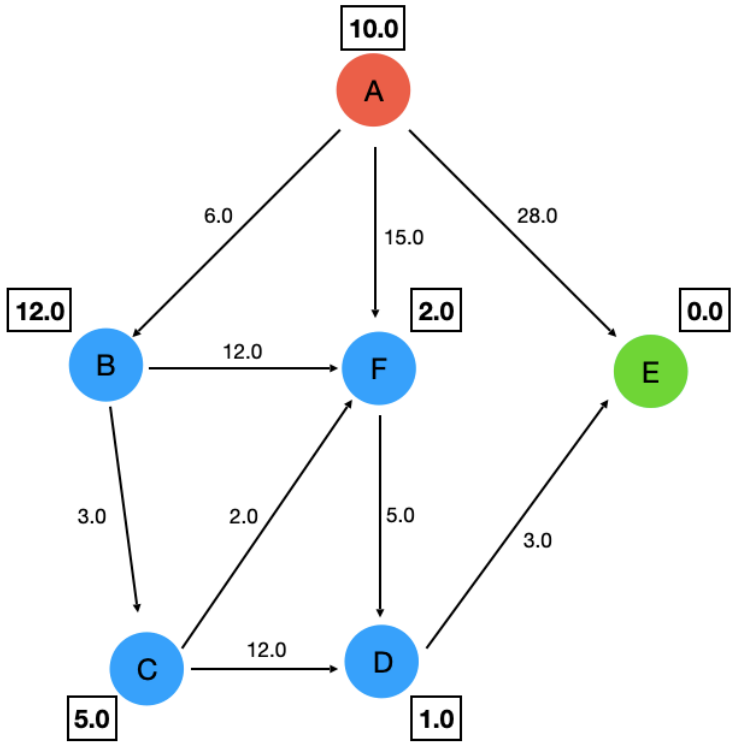
Exercise 1

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

Consider the following graph, where A is the initial node and E the goal node. The number on each arc is the cost of the operator for the move, while the number in the square next to each node is the heuristic evaluation of the node itself, namely, its estimated distance from the goal.



a) Apply the depth-first search (do not consider the costs of the nodes), and draw the developed search tree indicating the expansion order; in the case of non-determinism, choose the nodes to expand according to the alphabetical order. What is the produced solution and its cost?
b) Apply search A*, and draw the developed search tree indicating the expansion order and the value of the function f(n) for each node n. In the case of non-determinism, choose the nodes to expand according to the alphabetical order. Consider as heuristic h(n) the one indicated in the square next to each node in the figure. What is the produced solution and its cost?

Exercise 3

Given the following CSP:

A, B, C::[0..7]

A + 1 <= B
A + 4 >= C
B + 3 <= C

Apply the Arc-consistency to the CSP and show the final domains of the three variables.

Exercise 4

In the initial state described by the following atomic formulas:
[handempty, in(keys, pocket), in(letter, mailbox), entrance(door, home), compatible(keys, door)]

You want to reach the goal:
have(letter), inside(home)

The actions are modeled as follows:

grab(Object, Container)
PRECOND: handempty, in(Object, Container), ¬inside(home)
DELETE: handempty, in(Object, Container)
ADD: have(Object)

open(Entrance)
PRECOND: compatible(Key, Entrance), have(Key), ¬open(Entrance)
DELETE: have(Key)
ADD: handempty, open(Entrance)

get_inside(Place)
PRECOND: ¬inside(Place), open(Entrance), entrance(Entrance, Place)
DELETE: open(Entrance)
ADD: inside(Place)

Solve the problem with the POP algorithm, identifying threats and their solution during the process.

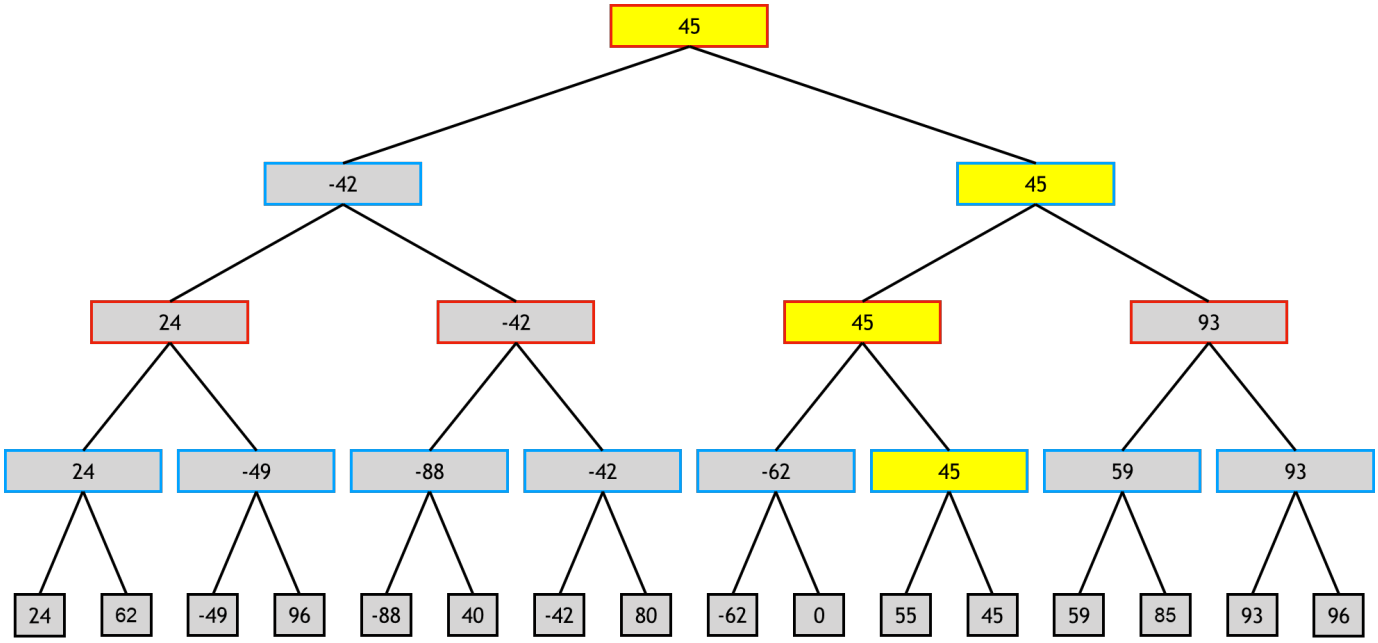
Exercise 5

1) Model the action **open** (preconditions, effects and frame axioms), and the initial state of the exercise 4 using the Kowalsky formulation.
2) Show two levels of graph plan when applied to exercise 4.
3) What is Breadth-First Search? Describe this search strategy and discuss its completeness and complexity.
4) What are the main features of a local search algorithm?
5) What are the main approaches of deductive planning. Explain the main differences.

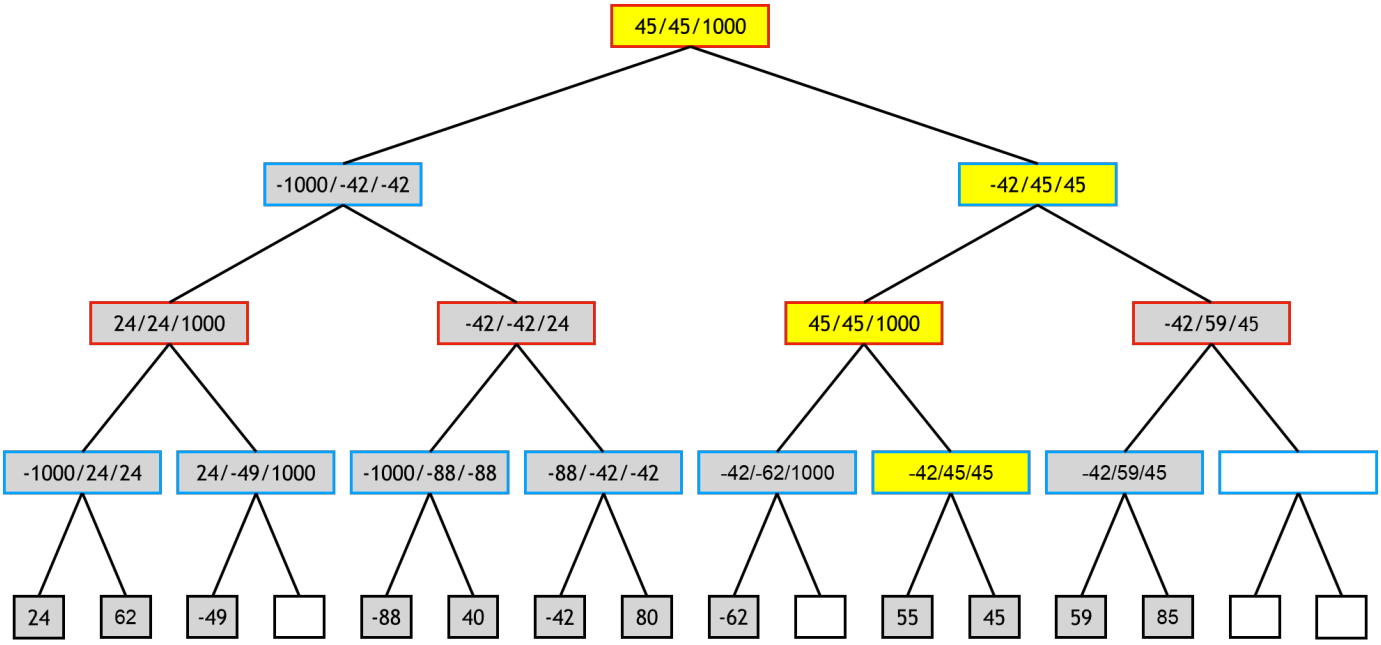
SOLUTION

Exercise 1

a) Min-max

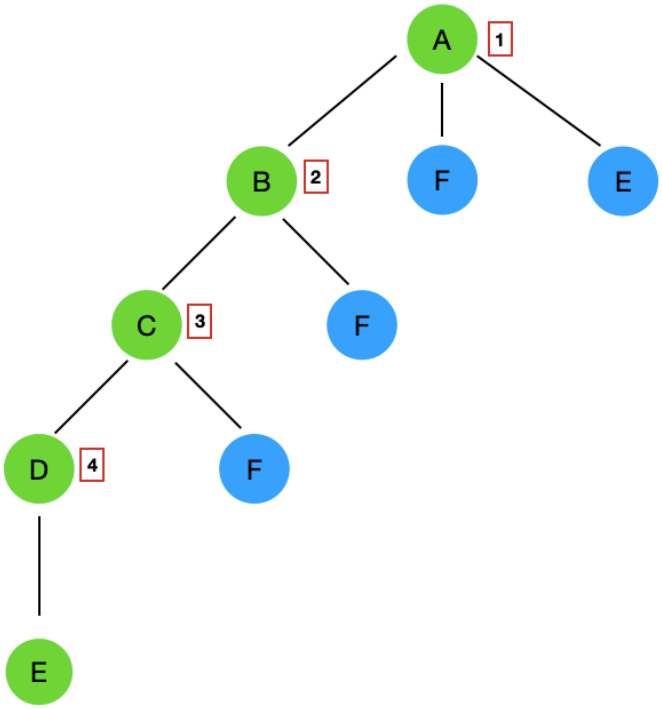


b) Alpha-beta cuts



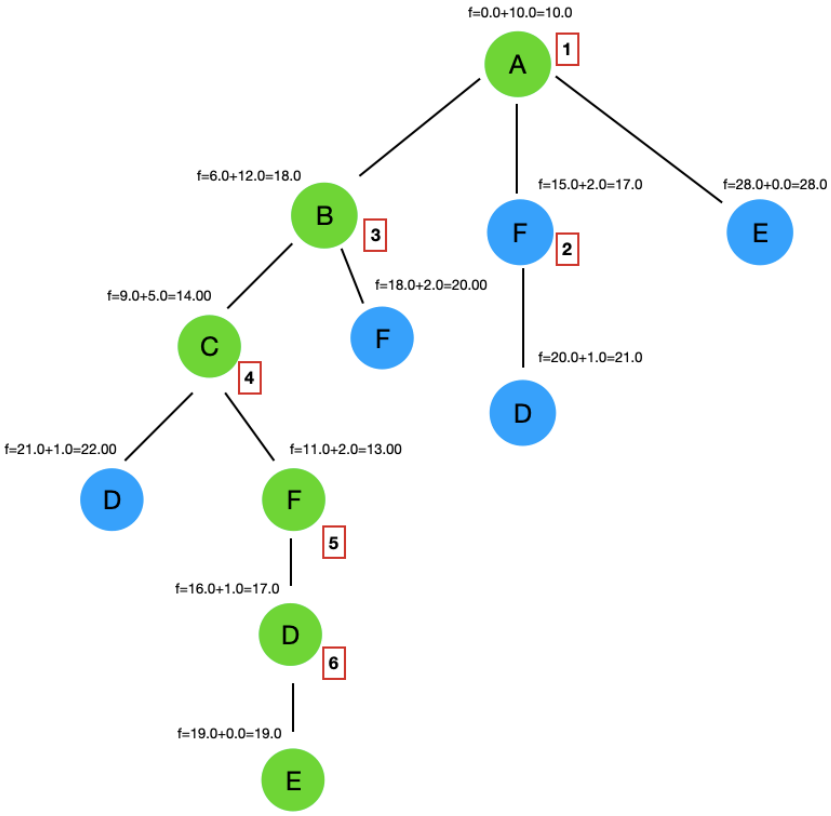
Exercise 2

a) Depth-first search



Solution: ABCDE with cost 24

b) A*



Solution: ABCFDE with cost 19

Exercise 3

By applying arc-consistency, the variables domains are reduced as follows:
A::[0..3]
B::[1..4]
C::[4..7]

***Applied reasoning*:**

We start with the following domains:
A::[0..7]
B::[0..7]
C::[0..7]

And with the following set of constraints:
a) A + 1 <= B
b) A + 4 >= C
c) B + 3 <= C

First Iteration:

Apply (a) **A + 1 <= B**

- For each A, is there a B? No. The only values of A which admit a solution are [0..6]
- For each B, is there an A? No. The only values of B which admit a solution are [1..7]

New domains:
A::[0..6]
B::[1..7]
C::[0..7]

Apply (b) **A + 4 >= C**

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

Domains not changed.

Apply (c) **B + 3 <= C**

- For each B, is there a C? No. The only values of B which admit a solution are [1..4]
- For each C, is there a B? No. The only values of C which admit a solution are [4..7]

New domains:
A::[0..6]
B::[1..4]
C::[4..7]

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

Apply (a) **A + 1 <= B**

- For each A, is there a B? No. The only values of A which admit a solution are [0..3]
- For each B, is there an A? Yes.

New domains:
A::[0..3]
B::[1..4]
C::[4..7]

Apply (b) **A + 4 >= C**

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

Domains not changed.

Apply (c) **B + 3 <= C**

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

Domains not changed.

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

Apply (a) **A + 1 <= B**

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

Domains not changed.

Apply (b) **A + 4 >= C**

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

Domains not changed.

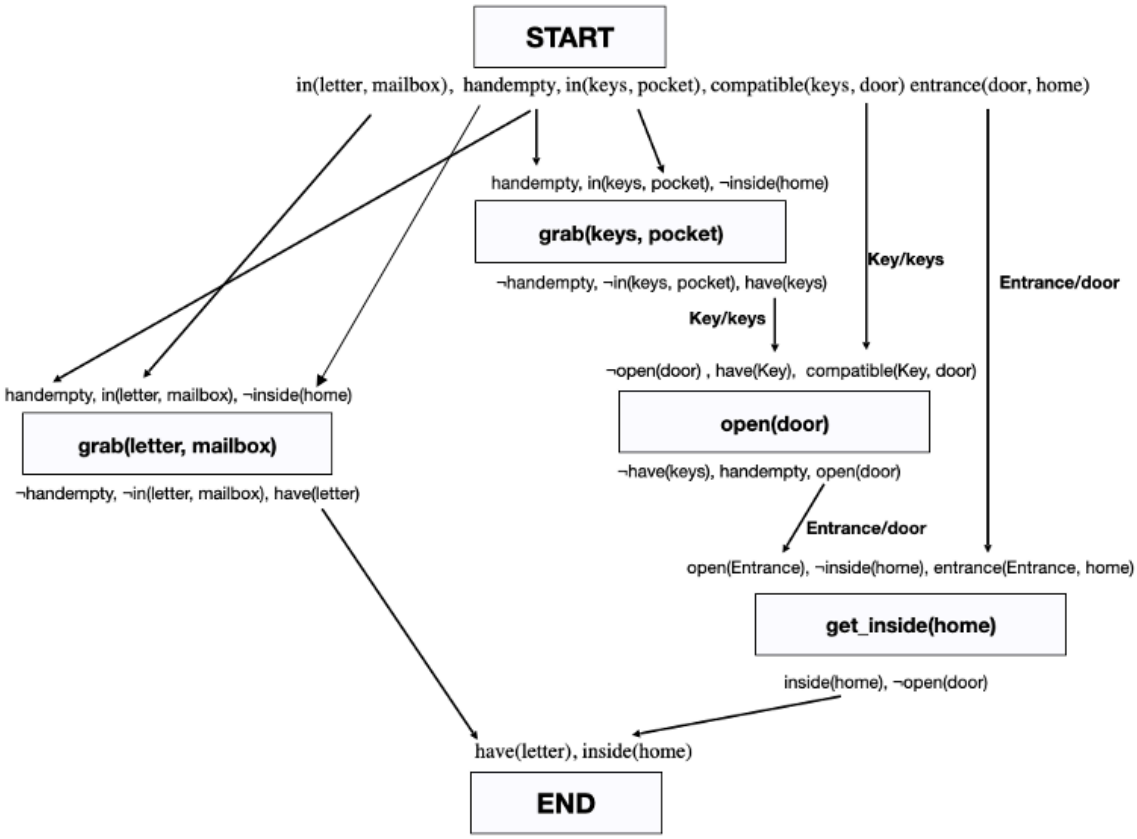
Apply (c) **B + 3 <= C**

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

Domains not changed.

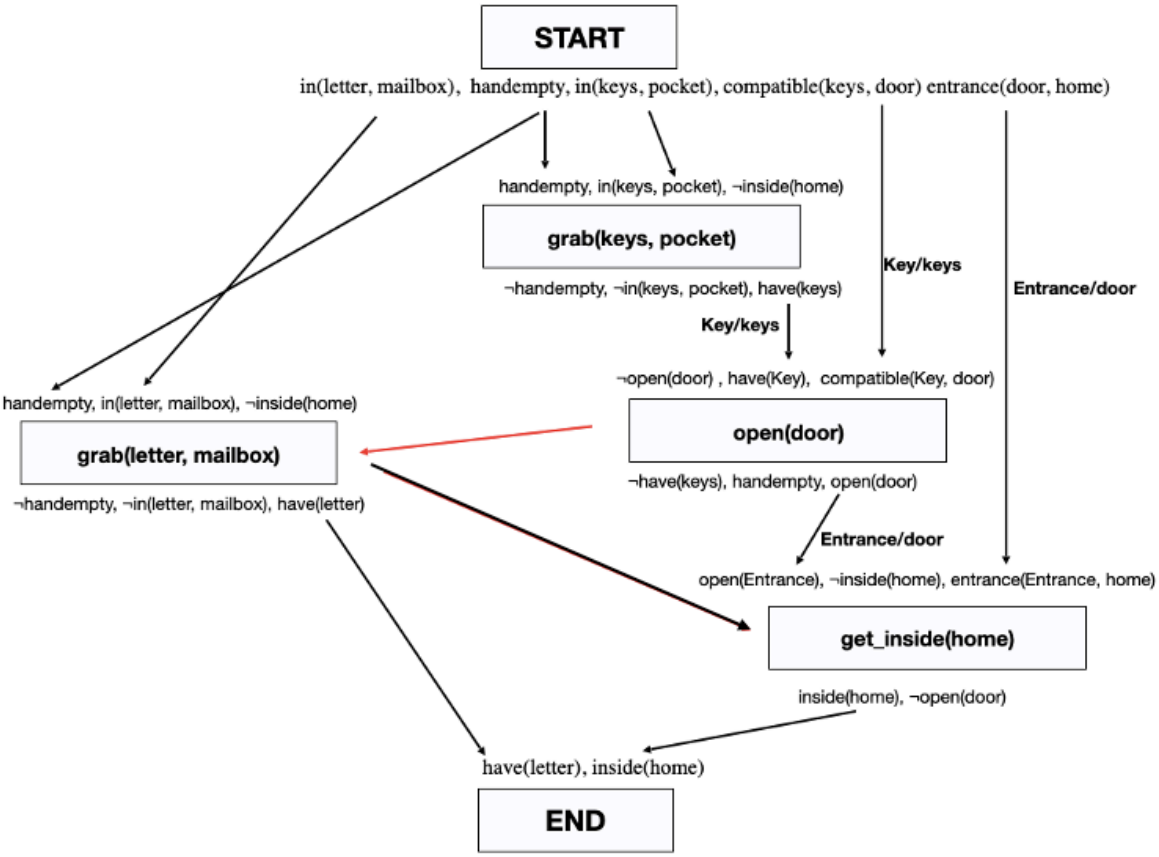
Algorithm termination.

Exercise 4



The plan created contains threats, in particular the causal links <Start, grab(letter, mailbox), handempty> and <Start, grab(keys, pocket), handempty> are threatened by the actions grab(letter, mailbox) and grab(keys, pocket), respectively.
A possible solution to the two conflicts is to add one ordering constraints, from the action grab(keys, pocket) and grab(letter, mailbox), then use the action open_door as a white knight for re-establishing handempty for the action grab(letter, mailbox).

At this point we have a threat caused by the action get_inside(home) and the causal link <Start, grab(letter, mailbox), ¬inside(home)> which can be solved by promotion.



Exercise 5

1) Kowalsky formulation

Initial State

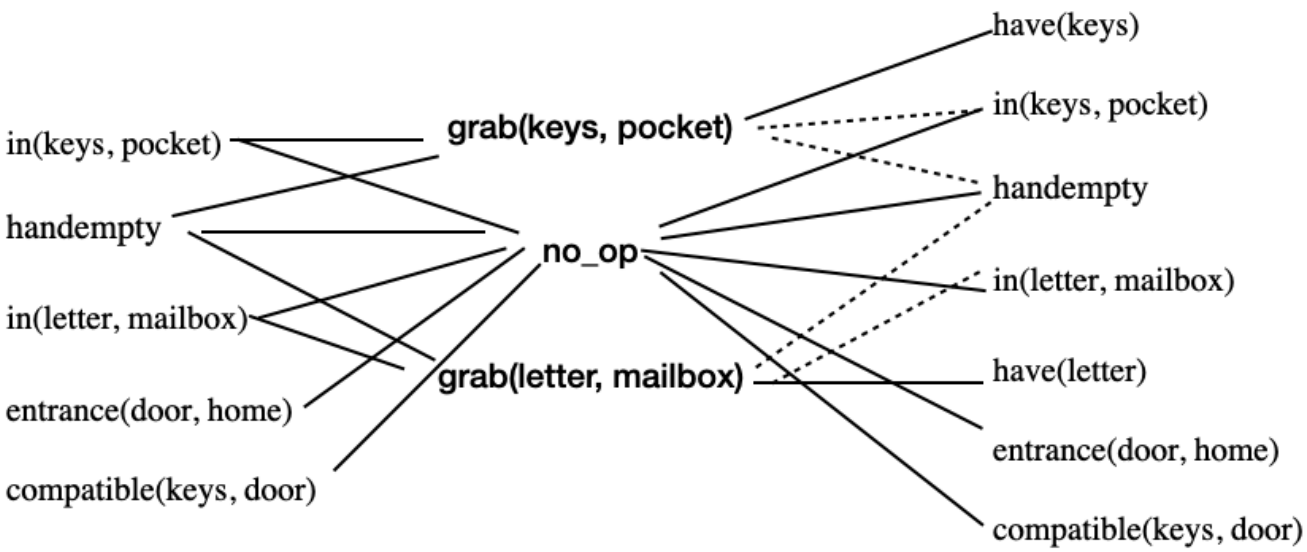
holds(in(letter, mailbox), s0).
holds(in(keys, pocket), s0).
holds(handempty, s0).
holds(entrance(door, home), s0).
holds(compatible(keys, door), s0)

Action open

holds(open(Entrance), do(open(Entrance), S)), holds(handempty, do(open(Entrance), S)).
pact(open(Entrance), S):- holds(have(Key), S), holds(¬open(Entrance), S), holds(compatible(Key, Entrance), S)
holds(V, do(open(Entrance), S)):- holds(V,S), holds(compatible(Key, Entrance), S), V\=have(Key).

The green predicate is needed to create a link between Entrance and Key.

2) Graphplan



no_op on **in(keys, pocket)** and **grab(keys, pocket)** are incompatible
no_op on **handempty** and **grab(keys, pocket)** are incompatible
no_op on **handempty** and **grab(letter, mailbox)** are incompatible
no_op on **in(letter, mailbox)** and **grab(letter, mailbox)** are incompatible
grab(keys, pocket) and **grab(letter, mailbox)** are incompatible

have (keys) and have(letter) are incompatible
in(keys, pocket) and have(keys) are incompatible
handempty and have(keys) are incompatible
have(letter) and in(letter, mailbox) are incompatible
have(letter) and have(keys) are incompatible