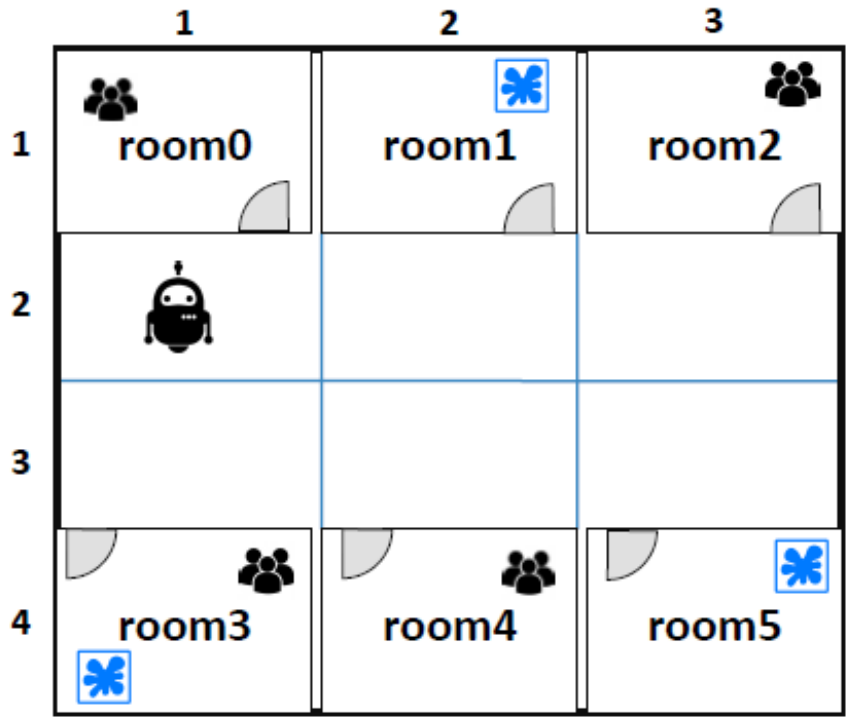


Hotel

(a) Describe the domain in PDDL

(b) Describe the problem in PDDL

(c) Discuss the forward planning process to reach the goal, using a perfect heuristic that gives for each state the number of steps to reach the goal; for each step, show the current state, the applicable actions and the state resulting from the application of the chosen action.

Our robot MARRtino works in a small hotel, whose map is represented as a grid world (see figure). MARRtino has the task of cleaning the rooms that are marked dirty in the map. MARRtino starts in the cell in front of room0 and can navigate the environment by moving inside the map in any of the 8 adjacent cells, that are traversable; it can also enter the hotel rooms by activating a specialized behavior, when is in the cell in front of the door. Once in a room MARRtino can clean it and then exit. MARRtino should not enter the rooms where it knows there are guests

Domain

```
(define (domain hotel-domain)
   (:requirements :strips)
   (:predicates (at ?x ?y) (adj ?x ?y) (wall ?y)
      (ro ?x) (busy ?x) (d ?x) (door ?x ?y))
   (:action move
      :parameters (?a ?from ?to)
      :precondition (and (at ?a ?from)
         (adj ?from ?to)
         (ro ?a)
         (not (door ?from ?to))
         (not (busy ?to)))
      :effect (and (not (at ?a ?from)) (at ?a ?to)))
   (:action clean
      :parameters (?a ?loc)
      :precondition (and (at ?a ?loc)
         (ro ?a)
         (d ?loc))
      :effect (and (not (d ?loc))))
   (:action traverse
      :parameters (?a ?from ?to)
      :precondition (and (at ?a ?from)
         (adj ?from ?to)
         (ro ?a)
         (door ?from ?to)
         (not (busy ?to)))
      :effect (and (not (at ?a ?from)) (at ?a ?to)))
)
```

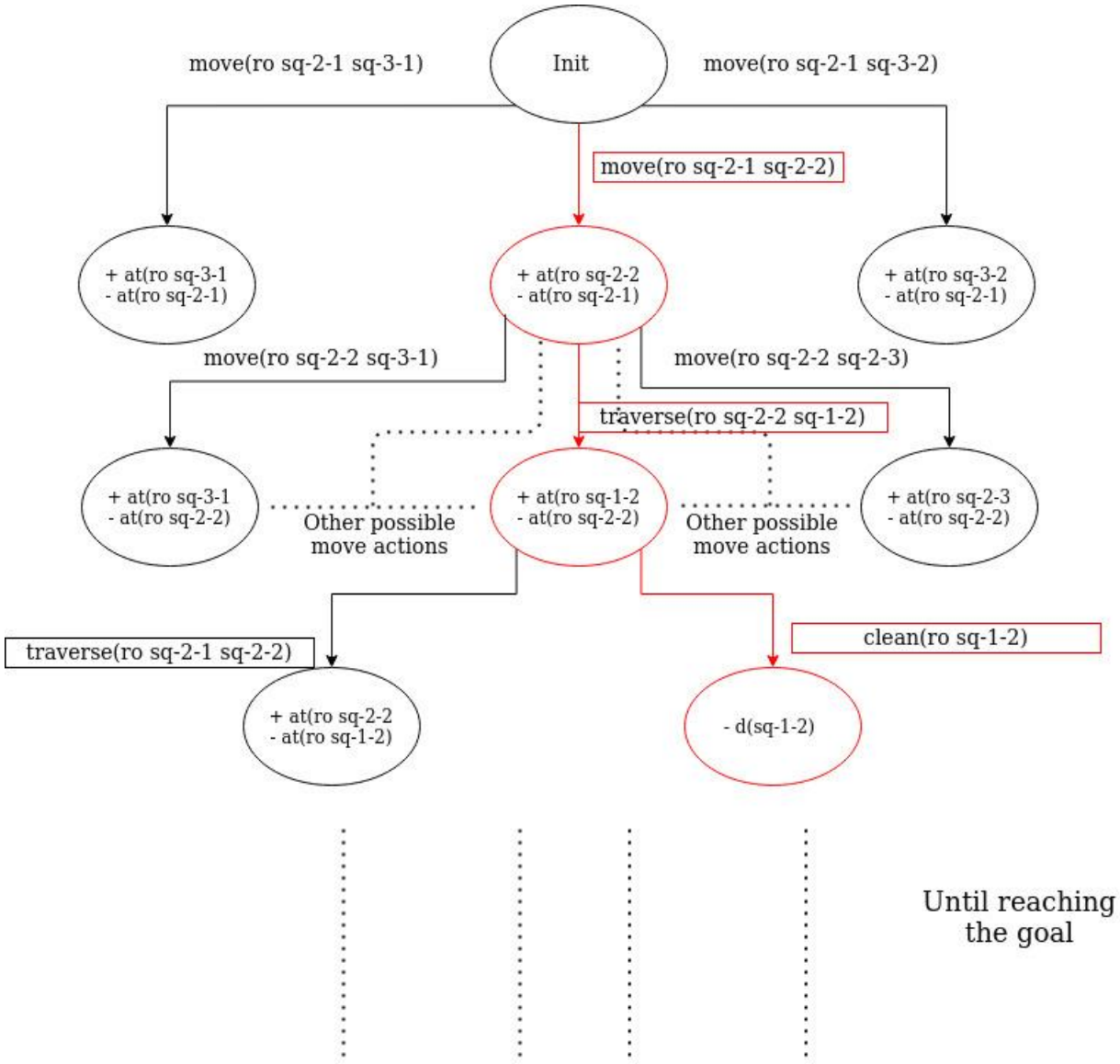
Problem (1)

```
(define (problem hotel-problem)
   (:domain hotel-domain)
   (:objects sq-1-1 sq-1-2 sq-1-3
   sq-2-1 sq-2-2 sq-2-3
   sq-3-1 sq-3-2 sq-3-3
   sq-4-1 sq-4-2 sq-4-3 robot)
   (:init (adj sq-1-1 sq-2-1) (adj sq-2-1 sq-1-1)
      (adj sq-1-2 sq-2-2) (adj sq-2-2 sq-1-2)
      (adj sq-2-1 sq-2-2) (adj sq-2-2 sq-2-1)
      (adj sq-1-3 sq-2-3) (adj sq-2-3 sq-1-3)
      (adj sq-2-1 sq-2-2) (adj sq-2-2 sq-2-1)
      (adj sq-2-1 sq-3-2) (adj sq-3-2 sq-2-1)
      (adj sq-2-1 sq-3-1) (adj sq-3-1 sq-2-1)
      (adj sq-2-2 sq-2-3) (adj sq-2-3 sq-2-2)
      (adj sq-2-2 sq-3-3) (adj sq-3-3 sq-2-2)
      (adj sq-2-2 sq-3-2) (adj sq-3-2 sq-2-2)
      (adj sq-2-2 sq-3-1) (adj sq-3-1 sq-2-2)
      (adj sq-2-3 sq-3-3) (adj sq-3-3 sq-2-3)
      (adj sq-2-3 sq-3-2) (adj sq-3-2 sq-2-3)
      (adj sq-3-1 sq-3-2) (adj sq-3-2 sq-3-1)
      (adj sq-3-1 sq-4-1) (adj sq-4-1 sq-3-1)
      (adj sq-3-2 sq-3-3) (adj sq-3-3 sq-3-2)
      (adj sq-3-2 sq-4-2) (adj sq-4-2 sq-3-2)
      (adj sq-3-3 sq-4-3) (adj sq-4-3 sq-3-3)
```

Problem (2)

```
(door sq-1-1 sq-2-1) (door sq-2-1 sq-1-1)
      (door sq-1-2 sq-2-2) (door sq-2-2 sq-1-2)
      (door sq-1-3 sq-2-3) (door sq-2-3 sq-1-3)
      (door sq-3-1 sq-4-1) (door sq-4-1 sq-3-1)
      (door sq-3-2 sq-4-2) (door sq-4-2 sq-3-2)
      (door sq-3-3 sq-4-3) (door sq-4-3 sq-3-3)
      (busy sq-1-1) (busy sq-1-3)
      (busy sq-4-1) (busy sq-4-2)
      (d sq-1-2) (d sq-4-1) (d sq-4-3)
      (at robot sq-2-1)
      (ro robot)
   )
   (:goal (and (not (d sq-1-2))
      (not (d sq-4-3)) ))
)
```

Point c


Init
move(ro sq-2-1 sq-3-1)
move(ro sq-2-1 sq-3-2)
move(ro sq-2-1 sq-2-2)
+ at(ro sq-3-1
- at(ro sq-2-1)
+ at(ro sq-2-2
- at(ro sq-2-1)
+ at(ro sq-3-2
- at(ro sq-2-1)
move(ro sq-2-2 sq-3-1)
move(ro sq-2-2 sq-2-3)
traverse(ro sq-2-2 sq-1-2)
+ at(ro sq-3-1
- at(ro sq-2-2)
Other possible
move actions
+ at(ro sq-1-2
- at(ro sq-2-2)
Other possible
move actions
+ at(ro sq-2-3
- at(ro sq-2-2)
traverse(ro sq-2-1 sq-2-2)
clean(ro sq-1-2)
+ at(ro sq-2-2
- at(ro sq-1-2)
- d(sq-1-2)
Until reaching
the goal