Homework2: Planning
Student: Ioana Veronica

1. I defined the following predicates:

Robot_full => #t if the robot has loaded 2 spheres, false

otherwise

Robot_empty => #t if the robot has loaded 0 spheres, false

otherwise

Robot_half => #t if the robot has loaded 1 sphere, false

otherwise

ColorsDif(n1, n2) => true is n1!= n2, false if the colors are equal

CanMove => true is robot full or robot_empty, false otherwise

CanLoad => true is robot_empty or robot_half, false otherwise

Move(room1, room2)

LP: Location(room1) ^ Door(room1, room2) ^ CanMove

LA:

Location(room2)

LE: :Location(room

1)

Load(color)

LP: Location(room) ^ Spheres(color, room, n) ^ Positive(n) ^ Color(room, roomColor) ^ ColorsDif(roomColor, color) ^ CanLoad

LA: Spheres(color, room, n1) ^ Succ(n1, n) ^ Carries(color, n2) ^ Succ

(n3, n2) LP: Carries(color, n3) ^ Spheres(color, room, n)

Unload(color)

LP: Location(room) ^ Carries(color, n) ^ Positive(n)

LA: Carries(color, n1) ^ Succ(n1, n) ^ Spheres(room, color, n2) ^

Succ(n3, n2) LE: Spheres(room, color, n3) ^ Carries(color, n)

2. For solving the second task of the homework I made a recursive function that takes into account the robot's position and how many balls it is carrying. Based on this information it identifies which case it is and based on the case it adds one or more moves to the reverse plan list. The robot starts moving from the initial position. Initially it will look for the room that holds the searched ball and it will move there. If we are not in the initial position and it is

carrying a ball it will look to see if it can take other balls with it from the final position and carry them out. The recursive function I built makes un update to the “world state” with each recall. This way if it adds a Load operation to the plan, it will modify the “world state”. After searching for the wanted ball, I call a function that finds the way to it and generates a series of moves of type “move” to it. On return, after loading the wanted ball it searches for the way back on the same principle as above. For memoryless-agent, the function that searches the way is not optimized.

3. For the third task of the homework, I modified a bit the former function, in order to simplify it a bit. Given the robot position and how many balls it is carrying, it can identify the whole plan that needs to be followed and it returns it. I optimized a bit the algorithm from task 2 in the following way: the robot doesn’t recalculate the “world state” at every step. The search algorithm (bsf) for the way to a certain location will not take any way, but the shortest way. It will perform a test every time before loading any balls.

At both the second and the third task, I used more auxiliary functions. There are comments in the source code for them.