CSD 311

ASSIGNMENT 4

Documentation - Goal Stack Planning

1. Pre-Conditions used -

We have displayed every intermediate stage of our project in terms of Predicate combinations. So the predicates that we have used for forming our states are:

1. **ON(X,Y) :** It means that X is stacked up over Y.
2. **ONTABLE(X) :** It means that there is no element under X, i.e X is lying on the table.
3. **CLEAR(X) :** It means that X is the topmost element, i.e nothing is present over X.
4. **HOLDING(X) :** It means that the robot's arm is holding X and it is yet to be placed somewhere.
5. **ARMEMPTY :** It means that the robot is not holding any block and so its arm is empty.

2. Algorithm / Approach :

We picked up the blocks one by one from every stack of the goal state and then compared it with the corresponding block of the initial state and then modified the latter block accordingly.

Suppose in the goal state A has been stacked up over B, whereas in initial state A is lying on the table and has C over it. So what our algorithm would do, it would check whether A is clear or not, and also check whether B is clear or not. If both are clear then the robot's arm would pick A and place it over B; else it would make adjustments and make B and A clear and then place A over B.

3. Functions used and their explanation -.

3.1 Function - ON(X,Y)

def on(block1,block2) :

- If block 1 is already piled up over block 2, then return; else,
- Remove any block present on block 2
- Hold block 1 in robot's arm
- And stack it over block 1
- now , remove block 2 from the clear list as it now has block 1 over it.
- Add block 1 to the clear list as it doesn't have anything on top.
- Add block 1 and block 2 in the block_on_block list.
- Empty the machine hand.

```
def on(blocks):
    global machine_hand, block_on_block, clear_blocks
    if blocks in block_on_block :
        return
    else:
        clear(blocks[1])
        holding(blocks[0])
        print("Stack", blocks[0], blocks[1])

        clear_blocks.remove(blocks[1])
        clear_blocks.add(blocks[0])
        block_on_block.add(blocks)
        blocks_on_table.remove(blocks[0])
        machine_hand = None
        print_state2()
```

3.2 Function - CLEAR(X)

def clear(block):

- This function takes a block parameter and then checks if there is another block present on it.
- If there is a block on it, then this function proceeds to remove the block present over it and make it clear.

- Pseudo Code -

if blockA has no another block B over it
  then return;
else
unstack all the blocks above the required designated blockA(using the unstack function);

```
def clear(block):
    global clear_blocks, block_on_block, blocks_on_table, machine_hand
    if block in clear_blocks:
        return
    else:
        unstack(block)
```

3.3 Function - ONTABLE(X)

def onTable(block):

- This function checks if the block is present on the table.
- If the block is not present on the table, then it unstacks all the blocks present above it (using the unstack function)
- And then uses the arm to place this block on the table
- Pseudo code -

if block on the table
 then return;
else
  unstack all blocks above it;
  place this block on the table;

```
def onTable(block):
    global clear_blocks, block_on_block, blocks_on_table, machine_hand
    if block in blocks_on_table:
        return
    else:
      unstack(block)
```

3.4 Function - HOLDING(X)

def holding(block):

- This function takes a block parameter and then checks if the machine arm is holding it.
- If there is no block being held by it or there is another block held by it, then this function proceeds to make the machine arm hold the desired block.
- Pseudo Code -

if machine arm holds the same block
  then return;
else
  remove the block held in the machine arm and place the desired block into it;

```
def holding(block):
    global machine_hand
    if machine_hand == block:
        return
    else:
        clear(block)
        onTable(block)
        armEmpty()

        machine_hand = block
        print("PickUp", machine_hand)
        print_state2()
```

3.5 ARMEMPTY()

def armEmpty():

- This function makes the robot's arm empty.
- If machine's hand is already empty, then return; else,
- Put down the block being holded by machine hand, and
- Add that block to blocks_on_table list, and
- Make the machine's hand empty.

```
def armEmpty():
    global machine_hand
    if machine_hand==None:
        return
    else:
        print("PutDown", machine_hand)
        blocks_on_table.add(machine_hand)
        machine_hand = None
        print_state2()
```

3.6 UNSTACK(X)

- This function takes a block as a parameter and finds out if any block sits on top of it. If yes then it unstacks the top block and puts the first block in a clear set.

```
def unstack(block):
    global clear_blocks, block_on_block, blocks_on_table, machine_hand
    a = block
    b = None
    for i in block_on_block:
        if a==i[1]:
            b = i[0]
            break
    if b==None:
        return
    clear(b)
    on((b,a))
    armEmpty()
    machine_hand = b
    clear_blocks.add(b)
    clear_blocks.add(a)
    block_on_block.remove((b,a))
    print("UnStack", b, a)
    print_state2()
```

3.7 The initial statements and printing loops -

```
blocks_on_table = set()
block_on_block = set()
clear_blocks = set()
machine_hand = None

for stack in initial_state:
    clear_blocks.add(stack[0])
    blocks_on_table.add(stack[-1])
    for block in range(len(stack)-1):
        block_on_block.add((stack[block],stack[block+1]))

def print_state(state):
    max = 0
    for i in state:
        if len(i) > max:
            max = len(i)
    temp = [[' '] * (max - len(x)) + x for x in state]
    for j in range(max):
        for i in range(len(temp)):
            print(temp[i][j], end = ' ')
        print()
    print('‾ ' * len(temp))

def print_initial_state():
    global initial_state
    print("Initial state")
    print_state(initial_state)

def print_goal_state():
    global goal_state
    print("Goal state")
    print_state(goal_state)

def print_globals():
    print(f'Clear = {clear_blocks}')
    print(f'Block on Block = {block_on_block}')
```

```
    print(f'On Table = {blocks_on_table}')
    print(f'Machine arm = {machine_hand}')

def print_state2():
    global blocks_on_table, block_on_block, clear_blocks, machine_hand
    state = ''
    for x in block_on_block:
        state += f"ON({x[0]},{x[1]}) ∧ "
    for x in blocks_on_table:
        state += f"ONTABLE({x}) ∧ "
    for x in clear_blocks:
        state += f"CLEAR({x}) ∧ "
    if machine_hand == None:
        state += "ARMEMPTY"
    else:
        state += f"HOLDING({machine_hand})"
    print(state)
    print()

    print(state)
    print()
```

3.8 Final running loop -

```
print_initial_state()
print_goal_state()
print_state2()

for i in range(2):
  for stack in goal_state:
      clear(stack[0])
      for block in range(len(stack)-2, -1, -1):
          on((stack[block], stack[block+1]))
      onTable(stack[-1])
      armEmpty()
```

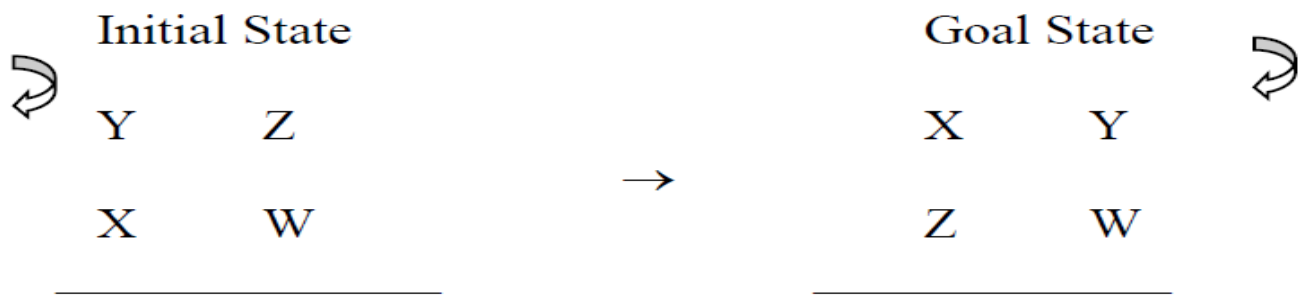
This loop outputs the initial and goal state at first and then runs the loop to check the goal state.

3. Outputs of test data -

3.1

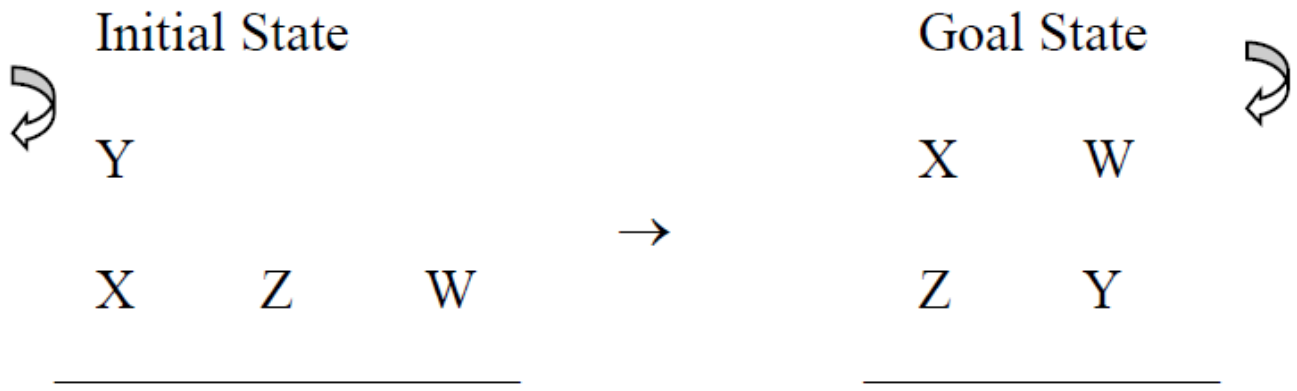

```
Initial state
Y W
X Z
- -

Goal state
Z Y
X W
- -

ON(Y,X) ∧ ON(W,Z) ∧ ONTABLE(Z) ∧ ONTABLE(X) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ ARMEMPTY

UnStack W Z
ON(Y,X) ∧ ONTABLE(Z) ∧ ONTABLE(X) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ HOLDING(W)

PutDown W
ON(Y,X) ∧ ONTABLE(Z) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ ARMEMPTY

UnStack Y X
ONTABLE(Z) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ CLEAR(X) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ HOLDING(Y)

PutDown Y
ONTABLE(Z) ∧ ONTABLE(X) ∧ ONTABLE(Y) ∧ ONTABLE(W) ∧ CLEAR(X) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ ARMEMPTY

PickUp Z
ONTABLE(Z) ∧ ONTABLE(X) ∧ ONTABLE(Y) ∧ ONTABLE(W) ∧ CLEAR(X) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ HOLDING(Z)

Stack Z X
ON(Z,X) ∧ ONTABLE(X) ∧ ONTABLE(Y) ∧ ONTABLE(W) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ ARMEMPTY

PickUp Y
ON(Z,X) ∧ ONTABLE(X) ∧ ONTABLE(Y) ∧ ONTABLE(W) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ HOLDING(Y)

Stack Y W
ON(Y,W) ∧ ON(Z,X) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ CLEAR(Z) ∧ CLEAR(Y) ∧ ARMEMPTY
```

3.2




```
Initial state
Y Z
X W
- -

Goal state
X Y
Z W
- -

ON(Z,W) ∧ ON(Y,X) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ ARMEMPTY

UnStack Y X
ON(Z,W) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ HOLDING(Y)

PutDown Y
ON(Z,W) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ ARMEMPTY

PickUp X
ON(Z,W) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ HOLDING(X)

Stack X Z
ON(Z,W) ∧ ON(X,Z) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ ARMEMPTY

UnStack X Z
ON(Z,W) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(Z) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ HOLDING(X)

PutDown X
ON(Z,W) ∧ ONTABLE(X) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(Z) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ ARMEMPTY

UnStack Z W
ONTABLE(X) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ CLEAR(X) ∧ HOLDING(Z)

PutDown Z
ONTABLE(W) ∧ ONTABLE(Y) ∧ ONTABLE(Z) ∧ ONTABLE(X) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ CLEAR(X) ∧ ARMEMPTY

PickUp Y
ONTABLE(W) ∧ ONTABLE(Y) ∧ ONTABLE(Z) ∧ ONTABLE(X) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ CLEAR(X) ∧ HOLDING(Y)

Stack Y W
ON(Y,W) ∧ ONTABLE(W) ∧ ONTABLE(Z) ∧ ONTABLE(X) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ CLEAR(X) ∧ ARMEMPTY

PickUp X
ON(Y,W) ∧ ONTABLE(W) ∧ ONTABLE(Z) ∧ ONTABLE(X) ∧ CLEAR(Y) ∧ CLEAR(Z) ∧ CLEAR(X) ∧ HOLDING(X)

Stack X Z
ON(X,Z) ∧ ON(Y,W) ∧ ONTABLE(W) ∧ ONTABLE(Z) ∧ CLEAR(Y) ∧ CLEAR(X) ∧ ARMEMPTY
```

3.3




```
Initial state
Y
X Z W
- - -

Goal state
X W
Z Y
- -

ON(Y,X) ∧ ONTABLE(Z) ∧ ONTABLE(W) ∧ ONTABLE(X) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(Y) ∧ ARMEMPTY

UnStack Y X
ONTABLE(Z) ∧ ONTABLE(W) ∧ ONTABLE(X) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ HOLDING(Y)

PutDown Y
ONTABLE(Z) ∧ ONTABLE(W) ∧ ONTABLE(X) ∧ ONTABLE(Y) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ ARMEMPTY

PickUp X
ONTABLE(Z) ∧ ONTABLE(W) ∧ ONTABLE(X) ∧ ONTABLE(Y) ∧ CLEAR(Z) ∧ CLEAR(W) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ HOLDING(X)

Stack X Z
ON(X,Z) ∧ ONTABLE(Z) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(W) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ ARMEMPTY

PickUp W
ON(X,Z) ∧ ONTABLE(Z) ∧ ONTABLE(W) ∧ ONTABLE(Y) ∧ CLEAR(W) ∧ CLEAR(X) ∧ CLEAR(Y) ∧ HOLDING(W)

Stack W Y
ON(W,Y) ∧ ON(X,Z) ∧ ONTABLE(Z) ∧ ONTABLE(Y) ∧ CLEAR(W) ∧ CLEAR(X) ∧ ARMEMPTY
```

3.4


```
Initial state
  E D
A C B
- - -

Goal state
  A D
E C B
- - -

ON(E,C) ∧ ON(D,B) ∧ ONTABLE(C) ∧ ONTABLE(A) ∧ ONTABLE(B) ∧ CLEAR(A) ∧ CLEAR(D) ∧ CLEAR(E) ∧ ARMEMPTY

UnStack E C
ON(D,B) ∧ ONTABLE(C) ∧ ONTABLE(A) ∧ ONTABLE(B) ∧ CLEAR(C) ∧ CLEAR(A) ∧ CLEAR(D) ∧ CLEAR(E) ∧ HOLDING(E)

PutDown E
ON(D,B) ∧ ONTABLE(C) ∧ ONTABLE(A) ∧ ONTABLE(B) ∧ ONTABLE(E) ∧ CLEAR(C) ∧ CLEAR(A) ∧ CLEAR(D) ∧ CLEAR(E) ∧ ARMEMPTY

PickUp A
ON(D,B) ∧ ONTABLE(C) ∧ ONTABLE(A) ∧ ONTABLE(B) ∧ ONTABLE(E) ∧ CLEAR(C) ∧ CLEAR(A) ∧ CLEAR(D) ∧ CLEAR(E) ∧ HOLDING(A)

Stack A C
ON(A,C) ∧ ON(D,B) ∧ ONTABLE(C) ∧ ONTABLE(B) ∧ ONTABLE(E) ∧ CLEAR(A) ∧ CLEAR(D) ∧ CLEAR(E) ∧ ARMEMPTY
```

4. Contributions -

Aman Yadav	Akansh Mittal	Aryaman Singh Rana