15-281 Spring 2023 Lecture Activity

Learning Objectives

- To define a classical planning problem with symbols and predicates
- To run BFS search, linear planning, and GraphPlan on the problem

A robot arm (yellow) can pick up and put down blocks to form stacks. It cannot pick up a block that has another block on top of it. It cannot pick up more than one block at a time. Any number of blocks can sit on the table.

Q1. Problem definition

(a) Suppose we have predicates: In-Hand(block), On-Table(block), On-Block(blockOver,blockUnder), Clear(block), and HandEmpty(). Write the instances in this problem.

Instances:
Blocks A, B, C

(b) Write the state of this configuration:

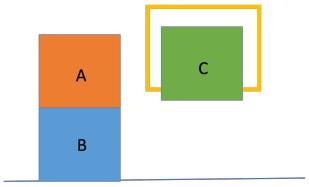


Figure 1: Block Configuration 1

State:
In-Hand(C) ∧ On-Table(B) ∧ On-Block(A,B) ∧ Clear(A) ∧ Clear(C)

(c) If instead we had the predicates: In-Hand(block), On(blockOver,loc), Clear(block), and Hand-Empty(). What would be the instances and state of Block Configuration 1 now?

Instances and State:
Blocks A, B, C, and Table. In-Hand(C) ∧ On(B,Table) ∧ On(A,B) ∧ Clear(A) ∧ Clear(C)

Q2. Operators

Use the instances and state from Q1a and Q1b to answer the following questions.

Define the following operators:

Pickup_Table(block):

Preconditions: HandEmpty() ∧ Clear(block) ∧ On-Table(block)

Add: In-Hand(block)

Delete: HandEmpty() ∧ On-Table(block)

Putdown_Table(block):

Preconditions: In-Hand(block)

Add: HandEmpty() ∧ On-Table(block)

Delete: In-Hand(block)

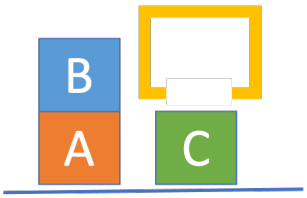
(a) Use the patterns above to write the operators Pickup_fromBlock(block,block) and Putdown_onBlock(block,block)

Pickup_fromBlock(block,fromBlock)
Pickup_fromBlock(block,fromBlock):
Precondition: HandEmpty(), On-Block(block,fromBlock), block != fromBlock
Add: In-Hand(block) ∧ Clear(fromBlock)
Delete: HandEmpty() ∧ On-Block(block,fromBlock)

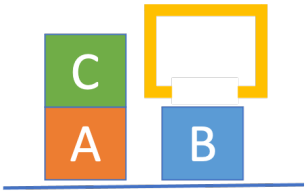
Putdown_onBlock(block,onBlock)
Putdown_onBlock(block,fromBlock):
Precondition: In-Hand(block), Clear(onBlock), block != onBlock
Add: HandEmpty() ∧ On-Block(block, onBlock)
Delete: In-Hand(block) ∧ Clear(onBlock)

Q3. Planning

Start State →



Goal State →



(a) Write the start state and goal state.

Start and Goal states
Start state: HandEmpty() ∧ On-Table(C) ∧ On-Table(A) ∧ On-Block(B, A) ∧ Clear(B) ∧ Clear(C)
Goal state: HandEmpty() ∧ On-Table(A) ∧ On-Table(B) ∧ On-Block(C, A) ∧ Clear(B) ∧ Clear(C)

(b) Perform BFS by matching operators to the current state, and adding an edge in the search tree for each matching operator. What is the plan found?

Plan
Pickup_fromBlock(B,A), Putdown_Table(B), Pickup_Table(C), Putdown_onBlock(C,A)

(c) Perform linear planning by solving for the goal in the following stack order (pop the rightmost item first): HandEmpty(), On-Table(A), On-Table(B), On-Block(C, A), Clear(B), Clear(C). What is the plan found?

Plan
Clear(C) - done (no operators)
Clear(B) - done (no operators)
On-Block(C,A) - Pickup_fromBlock(B,A), Putdown_Table(B), Pickup_Table(C), Putdown_onBlock(C,A)
On-Table(B) - done (no operators)
On-Table(A) - done (no operators)
HandEmpty() - done (no operators)
Final Plan: Pickup_fromBlock(B,A), Putdown_Table(B), Pickup_Table(C), Putdown_onBlock(C,A)

(d) What if we tried to achieve our goals in the opposite order (leftmost first): HandEmpty(), On-Table(A), On-Table(B), On-Block(C, A), Clear(B), Clear(C). What is the plan found?

Plan
HandEmpty() - done (no operators)
On-Table(A) - done (no operators)
On-Table(B) - Pickup_fromBlock(B,A), Putdown_Table(B)
On-Block(C,A) - Pickup_Table(C), Putdown_onBlock(C,A)
Clear(B) - done (no operators)
Clear(C) - done (no operators)
Final Plan: Pickup_fromBlock(B,A), Putdown_Table(B), Pickup_Table(C), Putdown_onBlock(C,A)

(e) Draw the first 2 layers of the GraphPlan graph.

Graphplan Graph

Q4. Challenge Problem: Word Puzzle

Here's a puzzle for you. You start with a 4 letter word, and there is a goal 4 letter word. You can take one action `change_one_letter` which transforms the word to a new valid word (cannot be made up). The goal is to determine what sequence of transformations changes the start word to the goal word the fastest.

For example: Suppose I'm given NECK and the goal is SOAP. I can perform the following actions:

$$NECK - DECK - DOCK - SOCK - SOAK - SOAP$$

You decide to model the problem using classical planning. You're given:

- a dictionary of words,
- a predicate `one_change(startword, endword)` which returns true if the `startword` and `endword` are 1 letter apart and false otherwise.
- a predicate `same_word(word1, word2)` which returns true if the two words are exactly the same and false otherwise.

Write an initial state and goal state that represents this problem. You might need to create other predicates.

Initial State:
Solution 1: `CURR(startword)` $\wedge(\forall word \neq startword,$ `!CURR(word))`
Solution 2: `CURR(startword)`

Goal State:
`CURR(goalword)`

Write an operator that makes use of the information above to find a sequence of words:

Operator:
If Solution 1:

```
change_one_letter:
    Preconditions:  CURR(word) AND one_change(word,nextword)
    Adds:  CURR(nextword), !CURR(word)
    Deletes:  CURR(word)
```

If Solution 2:

```
change_one_letter:
    Preconditions:  CURR(word) AND one_change(word,nextword)
    Adds:  CURR(nextword)
    Deletes:  CURR(word)
```