1 Vocabulary Check

Define each of the following terms:

1. Interference

   One action's effect deletes or negates a precondition of the other.

2. Inconsistent effects/Inconsistency

   One action's effect deletes or negates an effect of the other.

3. Competing Needs

   One action's precondition is the negation of a precondition of the other.

2 Compare and Contrast

1. What are some ways to find a plan using a classical planning environment model?

   Naive search (BFS), graph plan.

2. What classical planning assumptions are relaxed when using the GraphPlan heuristic? Why is this helpful compared to naive search?

   We are assuming we can take multiple non-mutex actions at the same time.

   This is useful since in this environment, taking multiple steps at a time will allow us to add multiple goals, finishing the search problem much quicker than the tradidional one action method

   (Also, if we return a plan that requires we take multiple actions at the same time, we can take them in any order with the same effect since they are non-mutex)

3 Journey to Success(or-State Axioms)

1. First, let's review some definitions. What are successor-state axioms?

   Successor-state axioms are axioms outlining what preconditions must be true in order to ensure that the state at the next time step will be specified. By definition, it is an axiom that sets the truth value of F^{t+1} (where F is some fluent, or changeable variable in an environment) in one of two ways:

   - The action at time t causes F to be true at $t+1$ (which refers to $ActionCausesF^t$)
   - F was already true at time t and the action at time t does not cause it to be false.

   It has the following schema: $F^{t+1} \iff ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$.
   We use successor-state axioms to ensure that each state we compute is the result of legal action.

2. Consider the following Mini Pacman grid. In this simplified world, the only available actions are *Left, Right,* and *Stay*. The only possible states are $Pacman_{(1,1)}$ and $Pacman_{(2,1)}$. If Pacman tries to move into a wall, he will stay in the same state.

   Notice that Pacman's state and actions are both fluent, so we can set up successor-state axioms to define how Pacman moves in this world. Write the successor-state axiom corresponding to Figure 4.

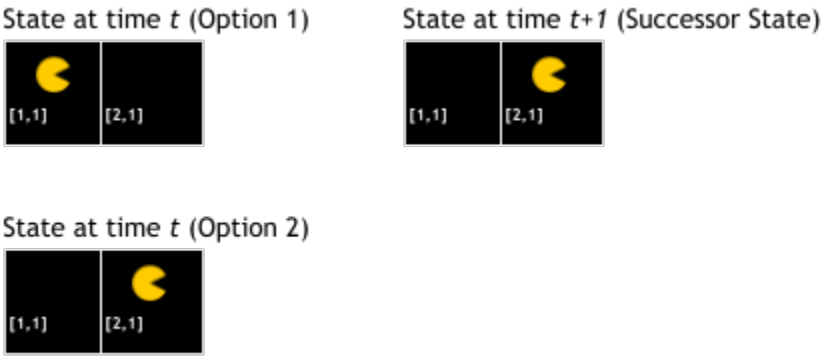


Figure 1: Mini Pacman Grid

Successor-state axiom: $Pacman^{t+1}_{(2,1)} \iff Right^t \vee (Pacman^t_{(2,1)} \wedge \neg Left^t)$

F^{t+1} is $Pacman^{t+1}_{(2,1)}$
$ActionCausesF^t$ is $Right^t$
$(F^t \wedge \neg ActionCausesNotF^t)$ is $(Pacman^t_{(2,1)} \wedge \neg Left^t)$

Think about how you could prevent Pacman from being in multiple states or taking multiple actions at the same time. You will get to explore this in P3!

3. Suppose that at time 0, Pacman is somewhere on a 5x5 grid ((1,1) at the bottom left, (5,5) at the top right) with only walls on the borders.

   For each of the following, state whether the entailment relation is correct. Explain your reasoning.

(a) $Up^t \vee Right^t \models \neg Pacman^{t+1}_{(1,1)}$

True, there is no square that would lead to square (1,1) after moving up or right

(b) $\neg Pacman^{t+1}_{(1,1)} \models Up^t \vee Right^t$

False, a counterexample would be starting at square (3,2), and an action left leading to square (2,2)

(c) $Up^0 \wedge Up^2 \wedge Up^3 \models Pacman^4_{(x,y)} : x \in [1,5], y \in [4,5]$

False, this is almost true, however, if Pacman starts at row 1 and the action at step 1 was down, Pacman would end at row 3

(d) $Up^t \wedge Right^t \models \neg Pacman^{t+1}_{(5,5)}$

True

There is no model that fits the action at a time step being both Up and Right. Therefore, for every model that fits this, the right side must also be true

(similar to being vacuously true for implications)

(e) $\neg Pacman^{t+1}_{(5,5)} \models Up^t \wedge Right^t$

False, since there is no model such that the right side is true, and there is at least one model such that the left side is true

(f) $Down^{t+1} \wedge Left^{t+1} \models Up^t \wedge Right^t$

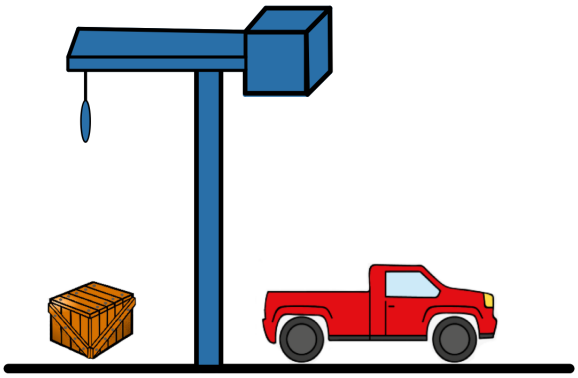
True, since there are no valid models in the left or the right

(similar to False $\implies$ False)

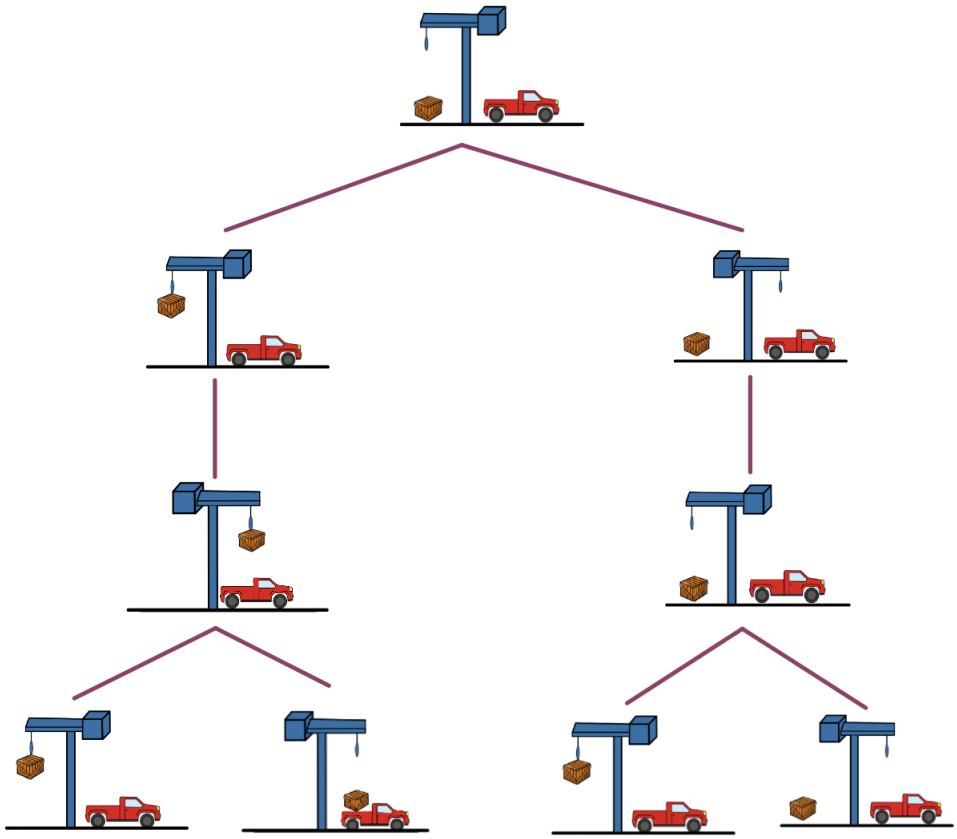
4 Symbolic Planning - Crate Problem

In the Crane problem, you are given a crane, a package and a truck. The package starts on the left, the truck on the right, and the crane faces the left. The goal of this is to load the package onto the truck and have the crane be facing the left.

The crane can swing between left and right, with or without a payload, and it can pick up the crate if it is on the same side. The crate can only be loaded onto the truck using the crane.



(a) Draw the planning graph for the first 3 moves. You may use pictures instead of propositions.

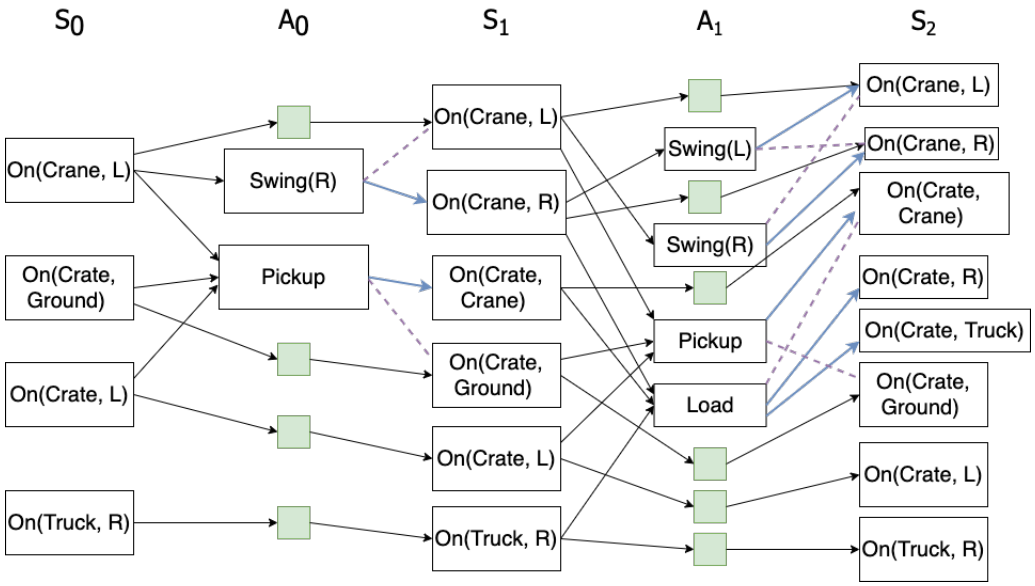


(b) Formulate the crate problem as a symbolic plan. You will need to define your variables, instances, start/goal states, and operators.

See provided sample code, bottom of the document

(c) Draw the first two levels of the Graph Plan graph.

In the following diagram, the blue lines represent the propositions added as the result of an action and the dotted purple lines represent the propositions deleted at the result of that action. The green squares in the action levels represent no-op's.



(d) Identify the exclusive actions in your graph and determine which type of mutex each is.

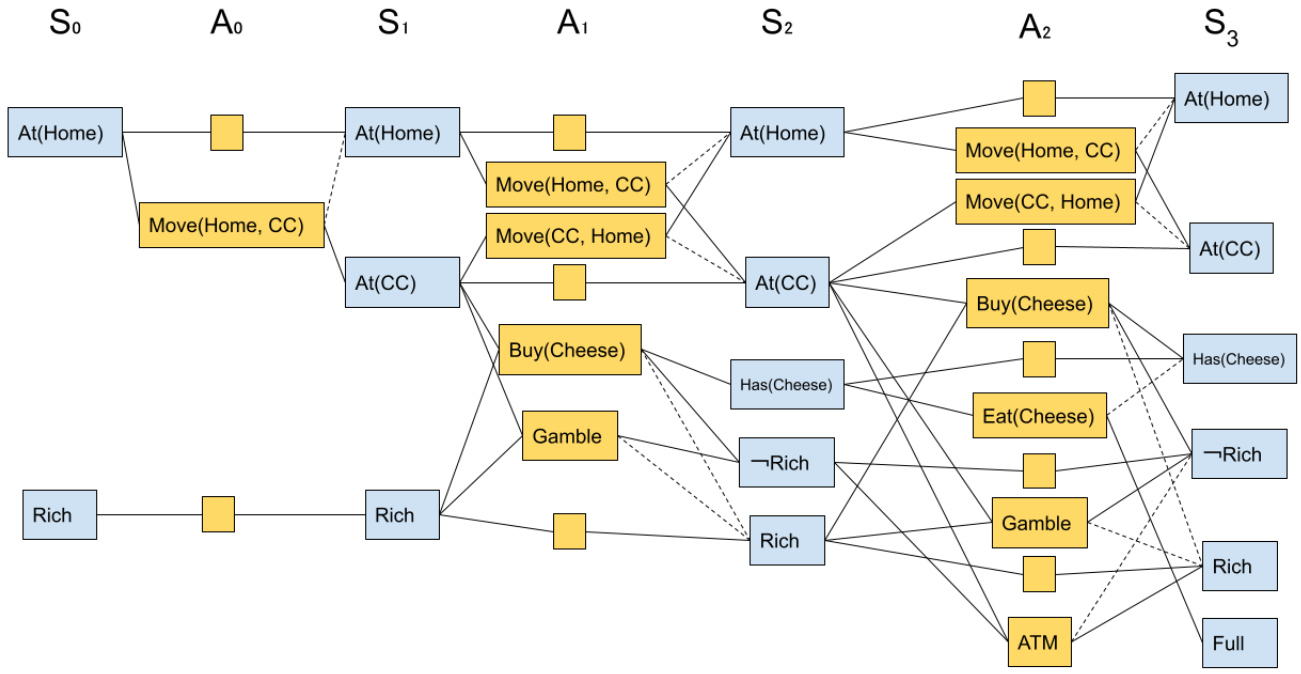
In the level A_0, Swing(R) and Pickup interfere with each other. In level A_1, one example would be Swing(L) and Swing(R) being inconsistent.

5 Mutex relation? I don't even know her!

Pinky is getting food from a Chuck E. Cheese. Pinky has the following actions:

- Move(A,B):
  - Preconditions: At(A)
  - Add list: At(B)
  - Delete list: At(A)
- Buy(Cheese):
  - Preconditions: At(ChuckyCheese), Rich
  - Add list: Has(Cheese), ¬ Rich
- Gamble
  - Preconditions: At(ChuckyCheese),Rich
  - Add list: ¬ Rich
  - Delete list: Rich
- ATM
  - Precondition: At(ChuckyCheese),¬ Rich
  - Add list: Rich
  - Delete list: ¬ Rich
- Eat(Cheese):
  - Preconditions: Has(Cheese)
  - Add list: Full
  - Delete list: Has(Cheese)

The start state contains the predicates Rich and At(Home).
The goal state is any state containing Full.
Below is the corresponding GraphPlan graph:



(a) Based on the above graph, list two actions that are mutex via inconsistent effects in level A_0.

No-op of At(Home) and Move(Home, ChuckyCheese)

(b) Based on the above graph, list two actions that are mutex via Interference in level A_1

Buy(Cheese) and Gamble()

(c) Based on the above graph, list two actions that are mutex via Competing needs in level A_2.

No-op of Rich and ATM

```
# Crane planning problem
from graphplanUtils import *

# Types
OBJ = 'Object'
DIR = 'Direction'
LOC = 'Location'

# Instances
truck = Instance('truck', LOC)
crane = Instance('crane', LOC)
crate = Instance('crate', OBJ)
ground = Instance('ground', LOC)

left = Instance('left', DIR)
right = Instance('right', DIR)

Instances = [truck, crane, crate, left, right, ground]

#Start and Goal States
Start = [Proposition('on', crane, left),
        Proposition('on', truck, right),
        Proposition('on', crate, ground),
        Proposition('on', crate, left)]

Goal = [Proposition('on', crane, left), Proposition('on', crate, truck)]


# Variables
v_to_side = Variable('to_side', DIR)
v_from_side = Variable('from_side', DIR)

# Operators
o_pickup = Operator('pickup',
        # Preconditions
        [Proposition('on', crate, ground),
        Proposition('on', crate, v_from_side),
        Proposition('on', crane, v_from_side)],
        # Adds
        [Proposition('on', crate, crane)],
        # Deletes
        [Proposition('on', crate, v_from_side),
        Proposition('on', crate, ground)])

o_swing = Operator('swing',
        # Preconditions
        [Proposition('on', crane, v_from_side),
        Proposition(NOT_EQUAL, v_from_side, v_to_side)],
        # Adds
        [Proposition('on', crane, v_to_side)],
        # Deletes
        [Proposition('on', crane, v_from_side)])

o_load = Operator('load',
```

```
        # Preconditions
        [Proposition('on', crane, v_from_side),
        Proposition('on', truck, v_from_side),
        Proposition('on', crate, crane)],
        # Adds
        [Proposition('on', crate, v_from_side),
        Proposition('on', crate, truck)],
        # Deletes
        [Proposition('on', crate, crane)])

Operators=[o_pickup, o_swing, o_load]

#Problems
prob1 = GraphPlanProblem('dockloading',
    # Instances
    Instances,
    # Operators
    Operators,
    # Initial state
    Start,
    # Goals
    Goal)

prob1.solve()
prob1.display()

# prob1.dump()
```