

# 1 Conceptual Review

1. Vocabulary check: Are you familiar with the following terms?

- Symbols:

Variables that can be T/F (capital letter)

- Operators:

And ( $\wedge$ ), Or ( $\vee$ ), Implies ( $\Rightarrow$ ), Equivalent ( $\Leftrightarrow$ )

- Sentences:

Symbols connected with operators, can be T/F

- Equivalence:

True in all models that  $A$  and  $B$  imply each other ( $A$  equivalent to  $B$ )

- Literals:

Atomic Sentence

- Knowledge Base:

Sentences known to be true

- Entailment:

$A$  entails  $B$  iff for every model that satisfies  $A$ ,  $B$  is also true

- Clauses (Definite vs. Horn Clauses):

**Clause:** A conjunction of literals

**Definite Clause:** Clause with exactly one positive literal

**Horn Clause:** Clause with at most one positive literal

- Model Checking:

Check if sentences are true in given model/check entailment

- Theorem Proving:

Search for sequence of proof steps (e.g. Forward Chaining)

- Modus Ponens:

From  $P$  and  $(P \Rightarrow Q)$ , infer  $Q$

- Linear Planning Algorithm

This algorithm greedily picks a subgoal and finds a plan for it. It does not move onto the next subgoal until it has found an plan for the current one. This algorithm is not necessarily complete or optimal, it usually works well with goals whose subgoals require mostly disjoint actions

- Non-Linear Planning

The agent interleaves actions to solve multiple subgoals at a time. This will be complete and optimal.

- Interference

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

- Inconsistent effects/Inconsistency

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

- Competing Needs

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

- Sussman's Anomaly

A reason for a linear planning algorithm to loop infinitely, this occurs when completing one subgoal undoes another subgoal, so the goal is never reached

2. Recall the definitions of satisfiability and entailment.

- **Satisfiability:**

A sentence is *satisfied by* some model (an assignment of values to variables)  $m$  if  $m$  makes the sentence true.

A sentence is satisfiable if there exists a model that satisfies it.

- **Entailment:**

Entailment:  $a \models b$  (" $a$  entails  $b$ " or " $b$  follows from  $a$ ") iff every model that satisfies  $a$  also satisfies  $b$ . In other words, the  $a$ -worlds (worlds where  $a$  is true) are a subset of the  $b$ -worlds [ $models(a) \subseteq models(b)$ ].

3. What is the difference between satisfiability and entailment?

Satisfiability holds if there exists a single model that in which the sentence is true. This is a property of a sentence (a sentence is satisfiable or it is not).

Entailment holds if all models which satisfy one sentence (query) also satisfy another sentence (knowledge base). It relates two sentences (sentence  $a$  entails  $b$ , or  $a$  does not entail  $b$ ), and requires checking all models that satisfy  $a$ .

4. Suppose  $A \models B$ . Consider all models assigning values to variables in sentences  $A$  and  $B$ . Which of the following sentences must be true in all possible models (even if either or both  $A/B$  are false)?

- |                       |                       |         |
|-----------------------|-----------------------|---------|
| (a) $A \wedge B$      | (c) $B \Rightarrow A$ | (e) $B$ |
| (b) $A \Rightarrow B$ | (d) $A \vee B$        |         |

(b)  $A \Rightarrow B$

By definition of implication, in all models (i.e., truth assignments) where  $A$  is true,  $B$  is also true. Thus in all models,  $A \Rightarrow B$  is satisfied.

(Thinking of it conversely, there would never be a model where  $B$  is false and  $A$  is true. By definition of implication, this means no model the rule  $A \Rightarrow B$ .)

5. Determine which of the following are correct, and explain your reasoning.

- $(A \vee B) \models (A \Rightarrow B)$

False (when  $A$  is true and  $B$  is False,  $A \vee B$  is true but  $A \Rightarrow B$  is false)

- $A \iff B \models A \vee \neg B$

True (the RHS is  $B \Rightarrow A$ , which is one of the conjuncts in the definition of  $A \iff B$ )

- $(A \vee B) \wedge \neg(A \Rightarrow B)$  is satisfiable

True (the model has  $A$  and  $\neg B$ )

6. How would we formulate the SAT problem as a CSP? What are the variables? Domains? Constraints?

SAT can be modeled as a CSP in which the variables are literals with domain  $\{\top, \perp\}$ , and the constraints are the clauses themselves.

7. Suppose we have an algorithm which determines whether a sentence is satisfiable or not. Given two sentences  $A$  and  $B$ , how could we determine whether  $A \models B$ ?

If  $A \models B$ , then  $A \wedge \neg B$  should be unsatisfiable (this is proof via reductio ad absurdum - reduction to an absurd thing).

8. What is the difference between linear and non-linear planning? When are they the same?

Linear planning: We keep a stack of unachieved goals and solve each one, one at a time, adding back goals that we violate along the way.

Non-Linear planning: Maintain a set of unachieved goals and search all interleavings of these goals adding a goal back to the set if a later change makes it violated.

Linear planning and non-linear planning are equivalent when there is one goal because there is only one possible "interleaving" of the goals so linear and non-linear planning will have the same approach.

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

Naive search (BFS), linear planning/non-linear planning, graph plan.

10. 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 traditional 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)

## 2 SATurdays are for everyone

1. Determine whether the sentences below are satisfiable or unsatisfiable (using any method you like).

(a)  $(\neg(Y \vee \neg Y) \vee X) \wedge (X \vee (Z \iff \neg Z))$

Satisfiable

**Logical reduction:**

$$\begin{array}{ll}
 (\neg(Y \vee \neg Y) \vee X) \wedge (X \vee (Z \iff \neg Z)) & \text{original sentence} \\
 ((\neg Y \wedge Y) \vee X) \wedge (X \vee (Z \iff \neg Z)) & \text{De Morgan's Law} \\
 (\perp \vee X) \wedge (X \vee (Z \iff \neg Z)) & (\neg Y \wedge Y) \text{ reduces to } \perp \\
 X \wedge (X \vee (Z \iff \neg Z)) & (\perp \vee X) \text{ reduces to } X \\
 X \wedge (X \vee ((Z \Rightarrow \neg Z) \wedge (\neg Z \Rightarrow Z))) & \text{Biconditional Elimination} \\
 X \wedge (X \vee ((\neg Z \vee \neg Z) \wedge (Z \vee Z))) & (\neg Z \vee \neg Z) \text{ reduces to } \neg Z, (Z \vee Z) \text{ reduces to } Z \\
 X \wedge (X \vee (\neg Z \wedge Z)) & (\neg Z \wedge Z) \text{ reduces to } \perp \\
 X \wedge (X \vee \perp) & (X \vee \perp) \text{ reduces to } X \\
 X \wedge X & (X \wedge X) \text{ reduces to } X \\
 X & 
 \end{array}$$

Assign  $X = \top$ . Then, we can choose any arbitrary assignment for  $Y$  and  $Z$ , and the sentence will be satisfied.

(b)  $\neg(X \vee \neg(X \wedge (Z \vee \top))) \implies \neg(Y \wedge (\neg Y \vee (\top \implies \perp)))$

Satisfiable

**Logical reduction:**

$$\begin{array}{ll}
 \neg(X \vee \neg(X \wedge (Z \vee \top))) \implies \neg(Y \wedge (\neg Y \vee (\top \implies \perp))) & \text{original sentence} \\
 (X \vee \neg(X \wedge (Z \vee \top))) \vee \neg(Y \wedge (\neg Y \vee (\top \implies \perp))) & \text{Implication Elimination} \\
 (X \vee \neg(X \wedge (Z \vee \top))) \vee \neg(Y \wedge (\neg Y \vee (\perp \vee \perp))) & \text{Implication Elimination} \\
 (X \vee \neg(X \wedge (Z \vee \top))) \vee \neg(Y \wedge (\neg Y \vee \perp)) & (\perp \vee \perp) \text{ reduces to } \perp \\
 (X \vee \neg(X \wedge (Z \vee \top))) \vee \neg(Y \wedge \neg Y) & (\neg Y \vee \perp) \text{ reduces to } \neg Y \\
 (X \vee \neg(X \wedge (Z \vee \top))) \vee \neg(\perp) & (Y \wedge \neg Y) \text{ reduces to } \perp \\
 (X \vee \neg(X \wedge (Z \vee \top))) \vee \top & \neg(\perp) \text{ reduces to } \top \\
 (X \vee \neg(X \wedge \top)) \vee \top & Z \vee \top \text{ reduces to } \top \\
 (X \vee \neg X) \vee \top & X \wedge \top \text{ reduces to } X \\
 \top \vee \top & X \vee \neg X \text{ reduces to } \top \\
 \top & \top \vee \top \text{ reduces to } \top
 \end{array}$$

We can choose any arbitrary assignment for  $X$ ,  $Y$  and  $Z$  and the sentence will be satisfied.

(c)  $((\top \iff \neg(X \vee \neg X)) \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \implies X))$

Unsatisfiable

**Logical reduction:**

$((\top \iff \neg(X \vee \neg X)) \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	<i>original sentence</i>
$((\top \iff \neg(\top)) \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	$(X \vee \neg X)$ reduces to $\top$
$((\top \iff \perp) \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	$\neg(\top)$ reduces to $\perp$
$((\top \Rightarrow \perp) \wedge (\perp \Rightarrow \top)) \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	<i>Biconditional Elimination</i>
$((\perp \vee \perp) \wedge (\top \vee \top)) \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	<i>Implication Elimination</i>
$((\perp \wedge \top) \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	$(\perp \vee \perp)$ reduces to $\perp$ , $(\top \vee \top)$ reduces to $\top$
$(\perp \vee Z) \vee Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	$(\perp \wedge \top)$ reduces to $\perp$
$Z \wedge \neg(Z \wedge ((Z \wedge \neg Z) \Rightarrow X))$	$((\perp \vee Z) \vee Z)$ reduces to $Z$
$Z \wedge \neg(Z \wedge (\perp \Rightarrow X))$	$(Z \wedge \neg Z)$ reduces to $\perp$
$Z \wedge \neg(Z \wedge (\top \vee X))$	<i>Implication Elimination</i>
$Z \wedge \neg(Z \wedge \top)$	$(\top \vee X)$ reduces to $\top$
$Z \wedge \neg Z$	$(Z \wedge \top)$ reduces to $Z$
$\perp$	$Z \wedge \neg Z$ reduces to $\perp$

This sentence is logically equivalent to  $\perp$ , and therefore cannot be satisfied.

### 3 Wandering in Wumpus World

We bring together what we have learned in lecture as well as the ideas of search so far in order to construct wumpus world agents that use propositional logic. The first step is to enable the agent to deduce, to the extent possible, the state of the world given its percept history. This requires writing down a complete logical model of the effects of actions. We also show how the agent can keep track of the world efficiently without going into the percept history for each inference. Finally, we show how the agent can use logical inference to construct plans that are guaranteed to achieve its goals.

Try it out: <http://thiagodnf.github.io/wumpus-world-simulator/> Note that there are some slight differences between this online version and the version we describe below.

Throughout this question, we will present several screenshots from the Wumpus World simulator linked previously. In each of these, assume that you *do* have an arrow on hand (as an extra exercise, consider how the answers might be different if you did not have an arrow). Also, note that the location of the explorer can be ignored. We just tried to place him somewhere where he wouldn't be blocking the text!

Recall that an agent in the Wumpus World has access to the following percepts:

- In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a Stench.
- In the squares directly adjacent to a pit, the agent will perceive a Breeze.
- In the square where the gold is, the agent will perceive a Glitter.
- When an agent walks into a wall, it will perceive a Bump.
- When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave.

1. Consider the following Wumpus World state:

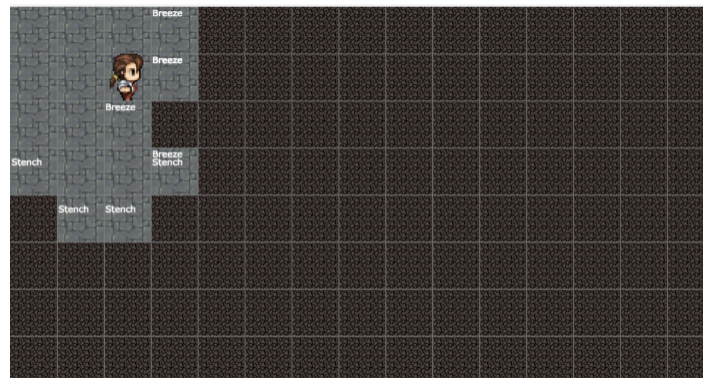
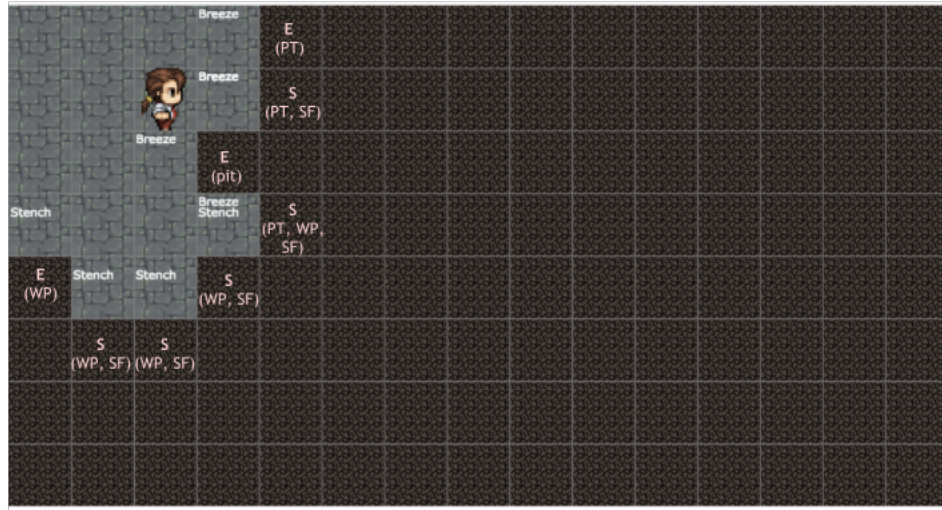


Figure 1: Entailment vs. Satisfiability?

Based on our previous discussion around entailment and satisfiability, identify locations where our knowledge base entails that there must be a Wumpus, Pit, or safe path. Additionally, identify locations where Wumpuses, Pit, and safe paths are not entailed but could be satisfied.

We've marked places where a pit (PT), wumpus (WP), safe (SF) can be satisfied with S-values. Entailments are indicated with E-values.



- Take a moment to familiarize yourself with the pseudocode below to understand how we might decide to act in Wumpus World. You'll notice that we have labeled the key decision-making portions of this code, and that different decisions need to be made given the state of our knowledge base.

On the next page, match each of the following states to one of the labeled code chunks in the pseudocode, and explain your reasoning.

**function** HYBRID-WUMPUS-AGENT(*percept*) **returns** an action  
**inputs:** *percept*, a list, [*stench*, *breeze*, *glitter*, *bump*, *scream*]  
**persistent:** *KB*, a knowledge base, initially the atemporal “wumpus physics”  
*t*, a counter, initially 0, indicating time  
*plan*, an action sequence, initially empty

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))  
TELL the *KB* the temporal “physics” sentences for time *t*  
*safe*  $\leftarrow \{[x, y] : \text{ASK}(\text{KB}, \text{OK}_{x,y}^t) = \text{true}\}$

<b>if</b> ASK( <i>KB</i> , <i>Glitter</i> <sup><i>t</i></sup> ) = true <b>then</b>	A
<i>plan</i> $\leftarrow$ [ <i>Grab</i> ] + PLAN-ROUTE( <i>current</i> , {[1,1]}, <i>safe</i> ) + [ <i>Climb</i> ]	
<b>if</b> <i>plan</i> is empty <b>then</b>	B
<i>unvisited</i> $\leftarrow \{[x, y] : \text{ASK}(\text{KB}, L_{x,y}^{t'}) = \text{false} \text{ for all } t' \leq t\}$	
<i>plan</i> $\leftarrow$ PLAN-ROUTE( <i>current</i> , <i>unvisited</i> $\cap$ <i>safe</i> , <i>safe</i> )	
<b>if</b> <i>plan</i> is empty and ASK( <i>KB</i> , <i>HaveArrow</i> <sup><i>t</i></sup> ) = true <b>then</b>	C
<i>possible_wumpus</i> $\leftarrow \{[x, y] : \text{ASK}(\text{KB}, \neg W_{x,y}) = \text{false}\}$	
<i>plan</i> $\leftarrow$ PLAN-SHOT( <i>current</i> , <i>possible_wumpus</i> , <i>safe</i> )	
<b>if</b> <i>plan</i> is empty <b>then</b> // no choice but to take a risk	D
<i>not_unsafe</i> $\leftarrow \{[x, y] : \text{ASK}(\text{KB}, \neg \text{OK}_{x,y}^t) = \text{false}\}$	
<i>plan</i> $\leftarrow$ PLAN-ROUTE( <i>current</i> , <i>unvisited</i> $\cap$ <i>not_unsafe</i> , <i>safe</i> )	
<b>if</b> <i>plan</i> is empty <b>then</b>	E
<i>plan</i> $\leftarrow$ PLAN-ROUTE( <i>current</i> , {[1, 1]}, <i>safe</i> ) + [ <i>Climb</i> ]	
<i>action</i> $\leftarrow$ POP( <i>plan</i> )	

TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))  
*t*  $\leftarrow$  *t* + 1  
**return** *action*

---

**function** PLAN-ROUTE(*current*, *goals*, *allowed*) **returns** an action sequence  
**inputs:** *current*, the agent’s current position  
*goals*, a set of squares; try to plan a route to one of them  
*allowed*, a set of squares that can form part of the route

*problem*  $\leftarrow$  ROUTE-PROBLEM(*current*, *goals*, *allowed*)  
**return** A\*-GRAPH-SEARCH(*problem*)

Figure 2: Hybrid-Wumpus-Agent from AIMA 3rd ed. It uses a propositional knowledge base to infer the state of the world, and a combination of problem-solving search and domain-specific code to decide what actions to take.




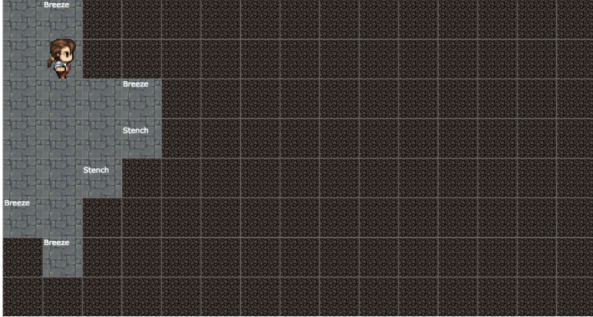

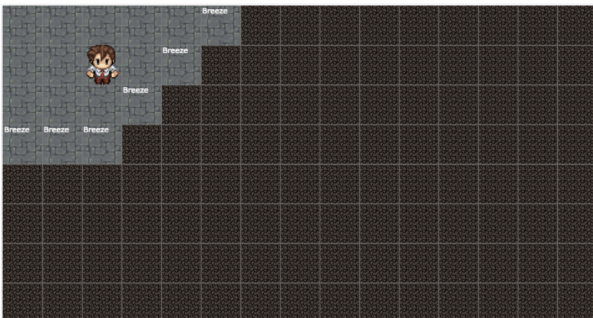
State	Code Chunk
	<p>C: There are two stenchs within reasonable distance; therefore, based on satisfiability, it is possible for a Wumpus to exist in the unvisited square diagonal from our explorer. There are also no guaranteed safe spaces. Since we have an arrow that we can use in case of Wumpus, we plan to shoot our arrow.</p>
	<p>B: We have found an unvisited square free from breeze or stench. We know, based on our knowledge base, that this unvisited square must therefore also be safe and we can visit it.</p>
	<p>A: We have found gold, and we can grab it, and then plan the shortest, safest route out.</p>
	<p>D: There is no guaranteed safe square; therefore, we must take a risk.</p>

Table 1: Which code chunk is applicable for each of these states?

## 4 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.

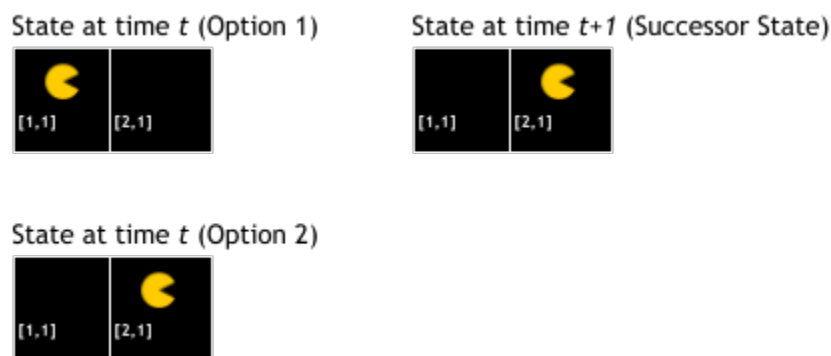


Figure 3: Mini Pacman Grid

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

$F^{t+1}$  is  $Pacman_{(2,1)}^{t+1}$

$ActionCausesF^t$  is  $Right^t$

$(F^t \wedge \neg ActionCausesNotF^t)$  is  $(Pacman_{(2,1)}^t \wedge \neg Left^t)$

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

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

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

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

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

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

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

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

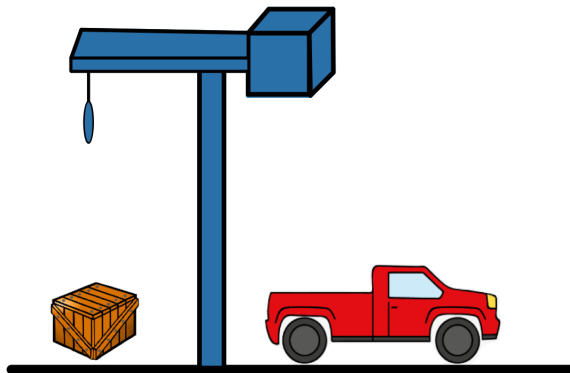
True, since there are no valid models in the left or the right

(similar to False  $\implies$  False)

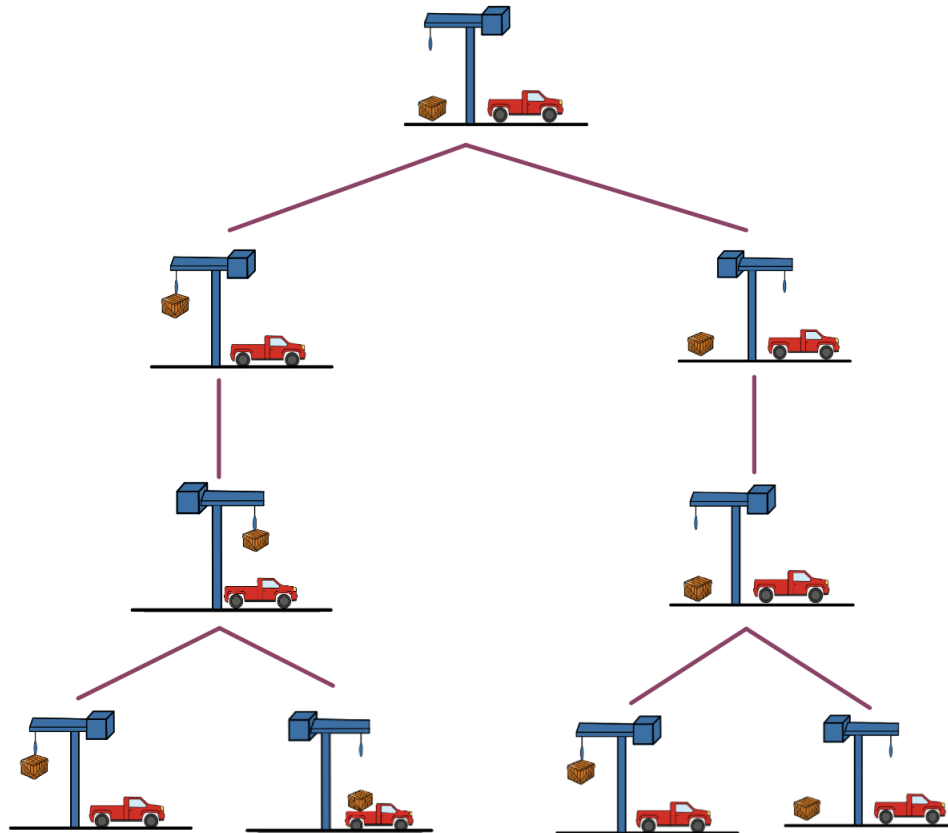
## 5 Symbolic Planning - Crate Problem

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.

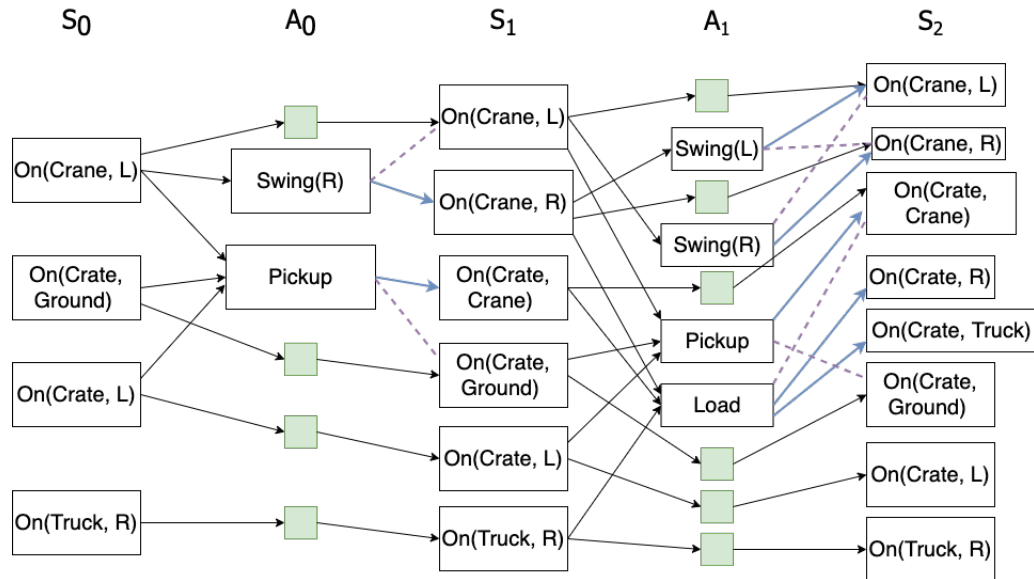


- (b) Formulate the crane problem as a symbolic plan. You will need to define your variables, instances, start/goal states, and operators.

[See provided sample code](#)

(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  $A_0$ ,  $\text{Swing(R)}$  and  $\text{Pickup}$  interfere with each other. In level  $A_1$ , one example would be  $\text{Swing(L)}$  and  $\text{Swing(R)}$  being inconsistent.

## 6 Mutex relation? I don't even know her!

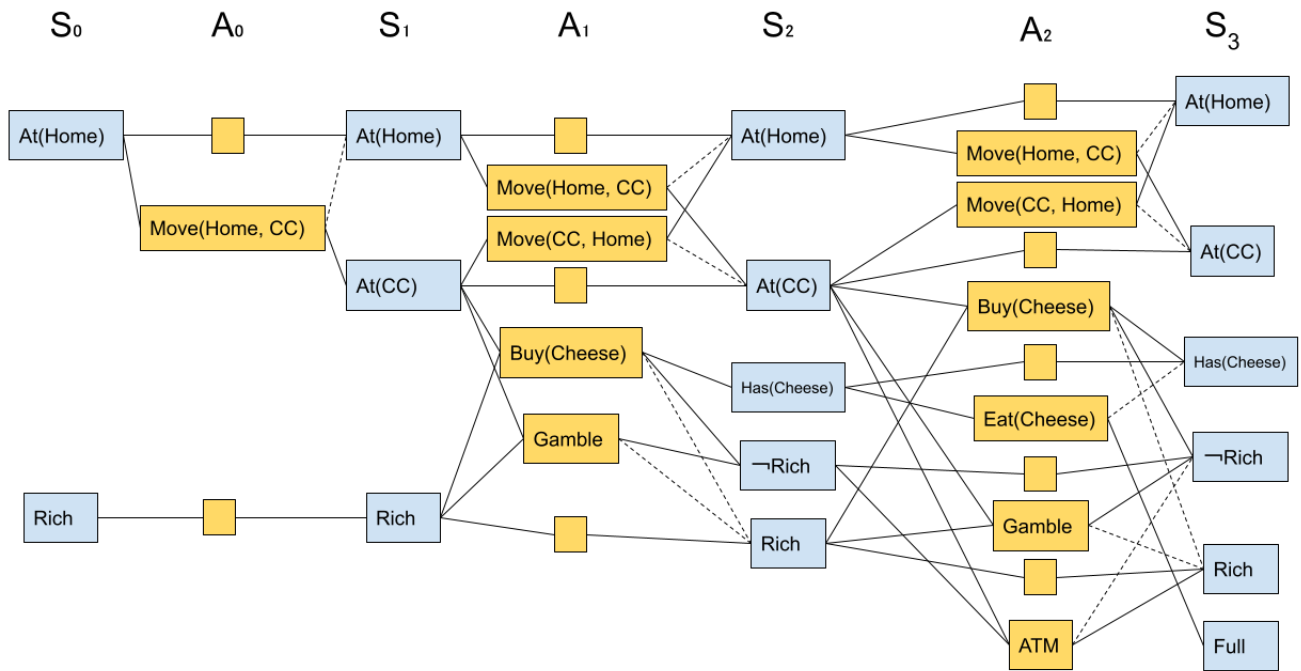
Pinky is getting food from a Chuck E. Cheese. Pinky has the following actions:

- Move(A,B):
  - Preconditions: At(A)
  - Add list: At(B)
  - Delete list: At(A)
- Buy(Cheese):
  - Preconditions: At(ChuckyCheese), Rich
  - Add list: Has(Cheese),  $\neg$  Rich
- Gamble
  - Preconditions: At(ChuckyCheese), Rich
  - Add list:  $\neg$  Rich
  - Delete list: Rich
- ATM
  - Precondition: At(ChuckyCheese),  $\neg$  Rich
  - Add list: Rich
  - Delete list:  $\neg$  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<sub>0</sub>.

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<sub>1</sub>

Buy(Cheese) and Gamble()

(c) Based on the above graph, list two actions that are mutex via Competing needs in level A<sub>2</sub>.

No-op of Rich and ATM