

1 Conceptual Review

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

- Symbols:
- Operators:
- Sentences:
- Equivalence:
- Literals:
- Knowledge Base:
- Entailment:
- Clauses (Definite vs. Horn Clauses):
- Model Checking:
- Theorem Proving:
- Modus Ponens:
- Linear Planning Algorithm
- Non-Linear Planning
- Interference

- Inconsistent effects/Inconsistency
- Competing Needs
- Sussman's Anomaly

2. Recall the definitions of satisfiability and entailment.

- **Satisfiability:**

- **Entailment:**

3. What is the difference between satisfiability and entailment?

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) $A \vee B$

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

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

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

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

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

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 $A \models B$?

8. What is the difference between linear and non-linear planning? When are they the same?
9. What are some ways to find a plan using a classical planning environment model?
10. What classical planning assumptions are relaxed when using the GraphPlan heuristic? Why is this helpful compared to naive search?

2 SATurdays are for everyone

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

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

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

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

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:

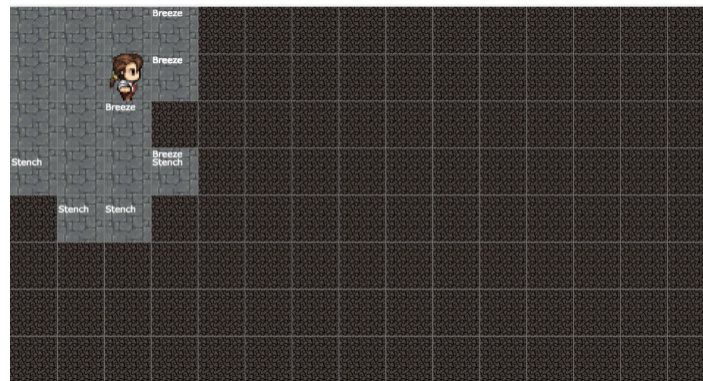


Figure 1: Entailment vs. Satisfiability?

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.

2. 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 $\leftarrow \{[x, y] : \text{ASK}(\text{KB}, \text{OK}_{x,y}^t) = \text{true}\}$

| | |
|---|----------|
| if ASK(<i>KB</i> , <i>Glitter</i> ^{<i>t</i>}) = true then <i>plan</i> \leftarrow [<i>Grab</i>] + PLAN-ROUTE(<i>current</i> , {[1,1]}, <i>safe</i>) + [<i>Climb</i>] | A |
| if <i>plan</i> is empty then <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 <i>plan</i> is empty and ASK(<i>KB</i> , <i>HaveArrow</i> ^{<i>t</i>}) = true then <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>) | C |
| if <i>plan</i> is empty then // no choice but to take a risk <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>) | D |
| if <i>plan</i> is empty then <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>) | E |

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.


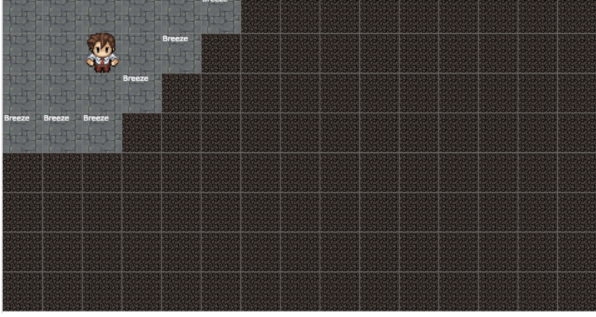
| State | Code Chunk |
|---|------------|
|  | |
|  | |
|  | |
|  | |

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?
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.

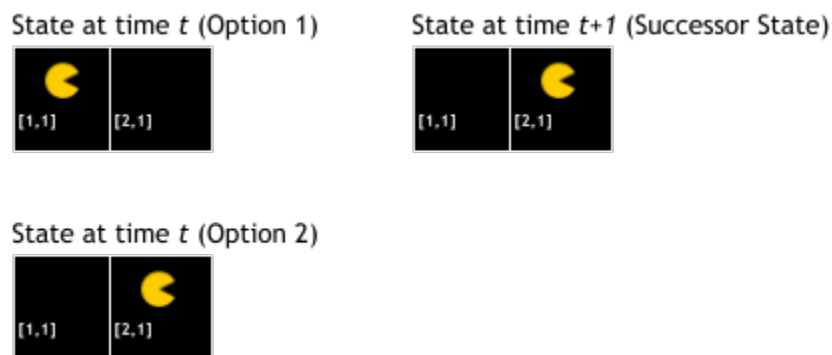


Figure 3: Mini Pacman Grid

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

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

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

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

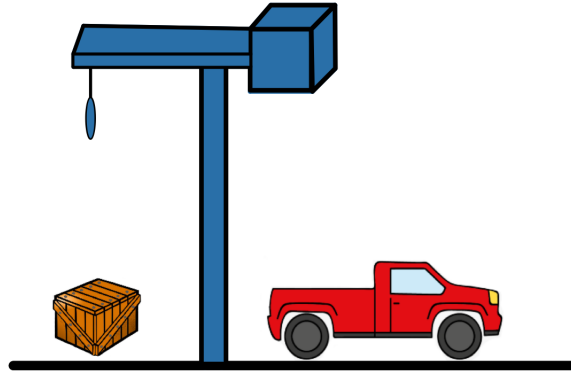
$$(e) \neg Pacman_{(5,5)}^{t+1} \models Up^t \wedge Right^t$$

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

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

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

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

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

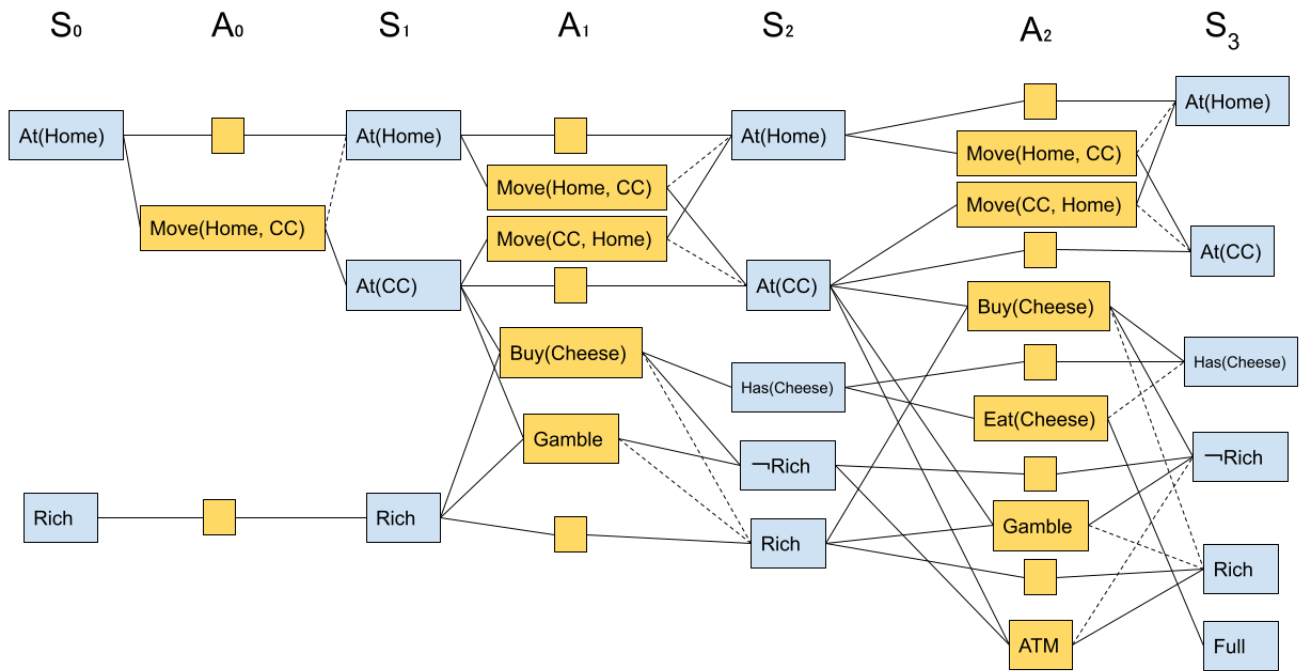
Pinky is getting food from a Chuck E. Cheese. Pinky has the following actions:

- 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:



- Based on the above graph, list two actions that are mutex via inconsistent effects in level A_0 .
- Based on the above graph, list two actions that are mutex via Interference in level A_1
- Based on the above graph, list two actions that are mutex via Competing needs in level A_2 .