

1 Conceptual Review

(a) Vocabulary check: Are you familiar with the following terms?

- Knowledge Base:
- Entailment:
- Clauses (Definite vs. Horn Clauses):
- Model Checking:
- Theorem Proving:
- Modus Ponens:

(b) Recall the definitions of satisfiability and entailment.

- **Satisfiability:**
- **Entailment:**

(c) What is the difference between satisfiability and entailment?

(d) 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 $A \models B$?

2 Forward Chaining

Consider the following forward chaining algorithm:

```

function ForwardChaining(KB, q) returns true or false
    count ← a table, where count[c] is the number of symbols in c's premise
    inferred ← a table, where inferred[s] is initially false for all s
    agenda ← a queue of symbols, initially symbols known to be true in KB
    while agenda is not empty do
        p ← Pop(agenda)
        if p = q then return true
        if inferred[p] = false then
            inferred[p] ← true
            for each clause c in KB where p is in c.premise do
                decrement count[c]
                if count[c] = 0 then add c.conclusion to agenda
    return false
  
```

(a) Consider the following propositional logic knowledge base that we will name KB1:

- | | |
|---|----------------------------------|
| • P | • $C_3 \wedge O \Rightarrow B_3$ |
| • $B_1 \wedge B_2 \wedge B_3 \Rightarrow G$ | • $W \Rightarrow E_1$ |
| • $D \Rightarrow G$ | • $W \Rightarrow E_2$ |
| • $E_1 \wedge S_1 \Rightarrow C_1$ | • $W \Rightarrow E_3$ |
| • $E_2 \wedge S_2 \Rightarrow C_2$ | • $P \Rightarrow S_1$ |
| • $E_3 \wedge S_3 \Rightarrow C_3$ | • $P \Rightarrow S_2$ |
| • $C_1 \wedge O \Rightarrow B_1$ | • $P \Rightarrow S_3$ |
| • $C_2 \wedge O \Rightarrow B_2$ | |

When calling `ForwardChaining(KB1, G)` with the above knowledge base and the query G , how many times is the body of the while loop in `ForwardChaining` executed? What does `ForwardChaining` return in this case?

(b) Suppose we now add W to the above knowledge base to get a new knowledge base, KB1_with_W:

- | | |
|---|------------------------------------|
| • W | • $E_1 \wedge S_1 \Rightarrow C_1$ |
| • P | • $E_2 \wedge S_2 \Rightarrow C_2$ |
| • $B_1 \wedge B_2 \wedge B_3 \Rightarrow G$ | • $E_3 \wedge S_3 \Rightarrow C_3$ |
| • $D \Rightarrow G$ | • $C_1 \wedge O \Rightarrow B_1$ |

- $C_2 \wedge O \Rightarrow B_2$
- $C_3 \wedge O \Rightarrow B_3$
- $W \Rightarrow E_1$
- $W \Rightarrow E_2$
- $W \Rightarrow E_3$
- $P \Rightarrow S_1$
- $P \Rightarrow S_2$
- $P \Rightarrow S_3$

When calling `ForwardChaining(KB1_with_W, G)` with this new knowledge base and the query G , how many times is the body of the while loop in `ForwardChaining` executed? What does `ForwardChaining` return in this case?

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. Note that the location of the explorer can be ignored. We just tried to place him somewhere off screen!

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.

(a) Consider the following Wumpus World state:

				Stench	H
A				Breeze Stench	G
B	Breeze	Breeze	Breeze	F	
	C	D	E		

Figure 1: Safe, not safe, or unsure?

For the squares A-H, mark each of them with '+' if the square is definitely safe, '-' if it is definitely not safe, and '?' otherwise.

- (b) Consider the PL symbols $S_A, S_B \dots S_H$ that represent whether or not each square A-H is safe or not safe. We also have symbols W_i and P_i for whether square i has a wumpus or a pit, respectively.

Let our Knowledge Base (KB) be a propositional logic sentence in CNF which contains all of the logic for the rules of Wumpus World as well as the percepts for the squares already visited (i.e., info about stench, breezes, or not in the 12 dark squares in the figure). Assume the KB also has logic related to the safety of a square, i.e., $\neg W_i \wedge \neg P_i \iff S_i$.

You are given access to a black box SAT solver function `PL_SATISFIES(pl_sentence)` which returns a PL model satisfying the given sentence *if* it is satisfiable and `None` otherwise.

Write pseudocode using the SAT solver and given propositional logic symbols to determine if square i is:

- (a) Definitely safe
 - (b) Definitely not safe
 - (c) Unsure
- (c) 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*
 $safe \leftarrow \{[x, y] : \text{ASK}(KB, OK_{x,y}^t) = \text{true}\}$

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