1 Discussion-Based Warm Ups

(a) Determine which of the following are correct, and explain your reasoning:

(i) $(A \vee B) \models (A \implies B)$

False, because when A is True and B is False, $A \vee B$ is true but $A \implies B$ is False.

(ii) $A \iff B \models A \vee \neg B$

True, because the right hand side is $B \implies A$, one of the conjuncts in the definition of $A \iff B$

(iii) $(A \vee B) \wedge \neg(A \implies B)$ is satisfiable.

True, model has A and $\neg B$.

(b) What is the difference between satisfiability and entailment (think about the purpose and requirements of each)?

From p. 250 in AIMA 3rd ed.: "A sentence is satisfiable if it is true in, or satisfied by, some model...

$\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable. Proving β from α by checking the unsatisfiability of $(\alpha \wedge \neg\beta)$ corresponds exactly to the standard mathematical proof technique of reductio ad absurdum (literally, reduction to an absurd thing).

Consider, for example, the agents location, initially $[1, 1]$, and suppose the agents unambitious goal is to be in $[2, 1]$ at time 1. The initial knowledge base contains $L^0_{1,1}$ and the goal is $L^1_{2,1}$. Now, SATPLAN will find the plan $[Forward^0]$; so far, so good. Unfortunately, SATPLAN also finds the plan $[Shoot^0]$. How could this be? To find out, we inspect the model that SATPLAN constructs: it includes the assignment $L^{2,1}_0$, that is, the agent can be in $[2, 1]$ at time 1 by being there at time 0 and shooting. One might ask, Didnt we say the agent is in $[1, 1]$ at time 0? Yes, we did, but we didnt tell the agent that it cant be in two places at once! For entailment, $L^{2,1}_0$ is unknown and cannot, therefore, be used in a proof; for satisfiability. Agents Based on Propositional Logic on the other hand, $L^{2,1}_0$ is unknown and can, therefore, be set to whatever value helps to make the goal true. For this reason, SATPLAN is a good debugging tool for knowledge bases because it reveals places where knowledge is missing. In this particular case, we can fix the knowledge base by asserting that, at each time step, the agent is in exactly one location. Satisfability is not necessarily guaranteed (exists in one of the models), but entailment is guaranteed.

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
  $safe \leftarrow \{[x, y] : \text{ASK}(KB, OK^t_{x,y}) = true\}$
  if ASK(KB, $Glitter^t$) = *true* **then** A
    plan ← [*Grab*] + PLAN-ROUTE(*current*, {[1,1]}, *safe*) + [*Climb*]
  if *plan* is empty **then** B
    $unvisited \leftarrow \{[x, y] : \text{ASK}(KB, L^{t'}_{x,y}) = false \text{ for all } t' \leq t\}$
    plan ← PLAN-ROUTE(*current*, *unvisited* ∩ *safe*, *safe*)
  if *plan* is empty and ASK(KB, $HaveArrow^t$) = *true* **then** C
    $possible\_wumpus \leftarrow \{[x, y] : \text{ASK}(KB, \neg\, W_{x,y}) = false\}$
    plan ← PLAN-SHOT(*current*, *possible_wumpus*, *safe*)
  if *plan* is empty **then** // no choice but to take a risk D
    $not\_unsafe \leftarrow \{[x, y] : \text{ASK}(KB, \neg\, OK^t_{x,y}) = false\}$
    plan ← PLAN-ROUTE(*current*, *unvisited* ∩ *not_unsafe*, *safe*)
  if *plan* is empty **then** E
    plan ← PLAN-ROUTE(*current*, {[1, 1]}, *safe*) + [*Climb*]
  action ← POP(*plan*)
  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 ← ROUTE-PROBLEM(*current*, *goals*,*allowed*)
  return A*-GRAPH-SEARCH(*problem*)

Figure 1: Hybrid-Wumpus-Agent from AIMIA 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.

2 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 back 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/

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!

(a) Consider the following Wumpus World state:

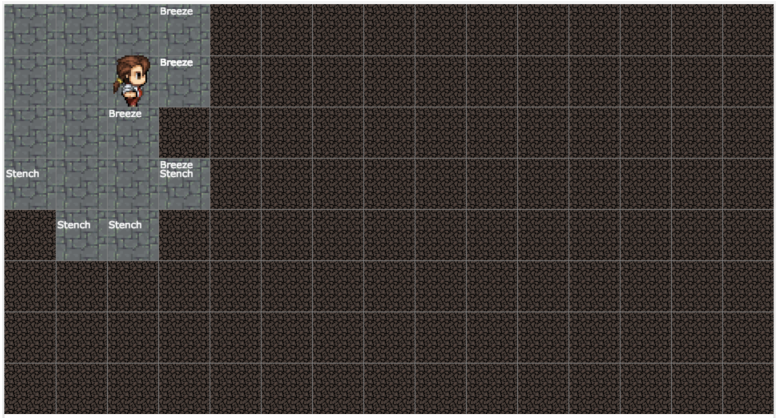
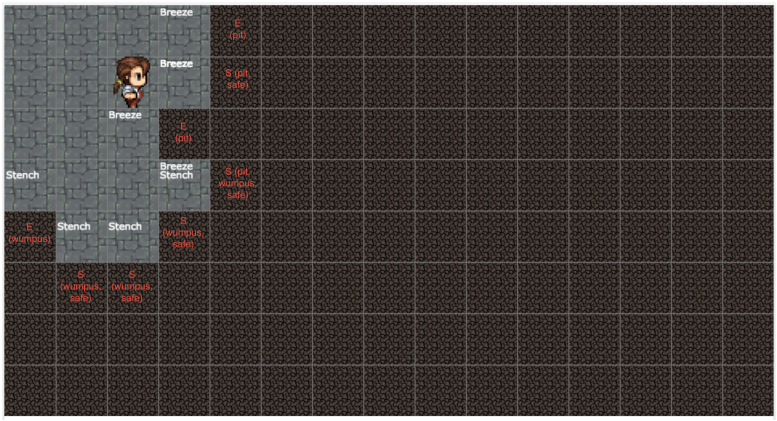


Figure 2: Entailment versus Satisfiability?

Based on our previous discussion around entailment and satisfiablity, 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, wampus, safe can be satisfied with S(type). Entailments are indicated with E(type).



(b) Now, refer to Figure 2 from Page 2, and take a moment to familiarize yourself with the pseudocode 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.

Match each of the following states to one of the labeled code chunks in the pseudocode, and explain your reasoning.

Figure 3: Which code chunk is applicable for each of these states?

1. C: There are two stenches 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. However, lucky for us, we have an arrow that we can use in case of Wumpus!
2. B: We have found a square free from breeze or stench that is adjacent to an unvisited square. We know, based on our knowledge base, that this unvisited square must therefore also be safe and can visit it.
3. A: We have found gold, and we can grab it, and then plan the shortest, safest route out.
4. D: There is no guaranteed safe square, therefore, we must take a risk.

3 Axioms & Arrows

Up until now we have assumed that the plans we create always make sure that an actions preconditions are satisfied. Let us now investigate what propositional successor-state axioms such as $HaveArrow^{t+1} \iff (HaveArrow^t \wedge \neg Shoot^t)$ have to say about actions whose preconditions are not satisfied.

(a) First, let us consider what successor-state axioms are. How do they differ from action axioms, and why might we choose to use them?

Action axioms give us the conditions needed at the current state for an action to be carried out at the current state. Successor state axioms are axioms outlining what preconditions in the current state need to be true in order to ensure that the state at the next timestep will be as 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. Also, writing only in terms of action axioms can become overwhelming, if there are a lot of time-dependent variables to keep track of. Successor state axioms are what help us do this.

(b) Show that the axioms predict that nothing will happen when an action is executed in a state where its preconditions are not satisfied.

We can illustrate the basic idea using the axiom given. Suppose that $Shoot^t$ is true but $HaveArrow^t$ is false. Then the RHS of the axiom is false, so $HaveArrow^{t+1}$ is false, as we would hope. More generally, if an action precondition is violated, then both $ActionCausesF^t$ and $ActionCausesNotF^t$ are false, so the generic successor-state axiom reduces to $F^{t+1} \iff False \vee (F^t \wedge True)$ which is the same as saying $F^{t+1} \iff F^t$, i.e., nothing happens.

(c) Consider a plan p that contains the actions required to achieve a goal but also includes illegal actions. Is it the case that successor-state axiom will allow the actions?

We recommend that you write a truth table and ask yourself the following question when looking at the truth table:

- Can I shoot if I don't have an arrow?

Yes, the plan plus the axioms will allow the actions and the axiom to evaluate to true; the axioms will copy every fluent across an illegal action and the rest of the plan will still work. Note that goal entailment is trivially achieved if we add precondition axioms, because then the plan is logically inconsistent with the axioms and every sentence is entailed by a contradiction. Precondition axioms are a way to prevent illegal actions in satisfiability-based planning methods (this will become clearer once we go over First Order Logic and Planning lectures). Refer to the truth table below where $A1$ refers to $HaveArrow^1$, $A2$ refers to $HaveArrow^2$ and $S1$ refers to $Shoot^1$.

A1	S1	A2	$A1 \wedge \neg S1$	$A2 \iff A1 \wedge \neg S1$	Reference
F	T	F	F	T	1
F	T	T	F	F	
T	T	F	F	T	
T	T	T	F	F	

Here, 1 refers to "Can I shoot if I don't have an arrow?"

4 SAT and DPLL

Recall the Davis-Putnam-Logemann-Loveland (DPLL) algorithm from lecture. DPLL conducts backtracking search over possible models (i.e., assignments to all variables) with these additional checks:

- Return true if all clauses are satisfied (we have found the answer)
- Return false if any clause is falsified (need to backtrack).
- If all occurences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value; e.g., in $(\neg A \vee B) \wedge (\neg A \vee \neg C)$, set A to false, which satisfies all the remaining clauses.
- If a clause is left with a single literal, set the literal to satisfy the clause; e.g., in $(B) \wedge (\neg B \vee \neg C)$, set B to true to satisfy the left clause. This often leads to new unit clauses, as in this example where $\neg C$ will be in a clause by itself.

(a) What is the difference between truth table entailment and solving SAT with DPLL?

For a knowledge base to entail a sentence α, α must be true in ALL models that are consistent with the knowledge base.

However, for a sentence to be satisfiable, there only needs to be at least one model for which the sentence is true.

(b) Find if the following is satisfiable. Other than the above rules, assign values to symbols alphabetically, and begin with symbol = true.

$(\neg A \vee C) \wedge (B \vee D) \wedge (A \vee \neg C) \wedge (\neg B \vee C) \wedge (B \vee \neg C)$

All occurrences of D are positive, so assign D = true. This leaves

$(\neg A \vee C) \wedge (true) \wedge (A \vee \neg C) \wedge (\neg B \vee C) \wedge (B \vee \neg C)$

There are no symbols for which the above rules apply, so begin with A = true. This leaves

$(C) \wedge (true) \wedge (true) \wedge (\neg B \vee C) \wedge (B \vee \neg C)$

C is the only literal in the first clause, so set C = true. This leaves

$(true) \wedge (true) \wedge (true) \wedge (true) \wedge (B)$

B is the only literal in the last clause, so set B = true. This leaves

$(true) \wedge (true) \wedge (true) \wedge (true) \wedge (true)$.

Since all clauses are true, we return true. The final assignments were A = true, B = true, C = true, D = true (although these are not returned by DPLL; only the boolean SAT is returned).

5 Resolution

Resolution

Algorithm Overview

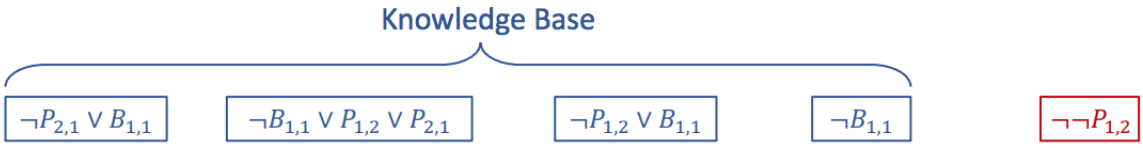
function PL-RESOLUTION?(KB, α) returns true or false

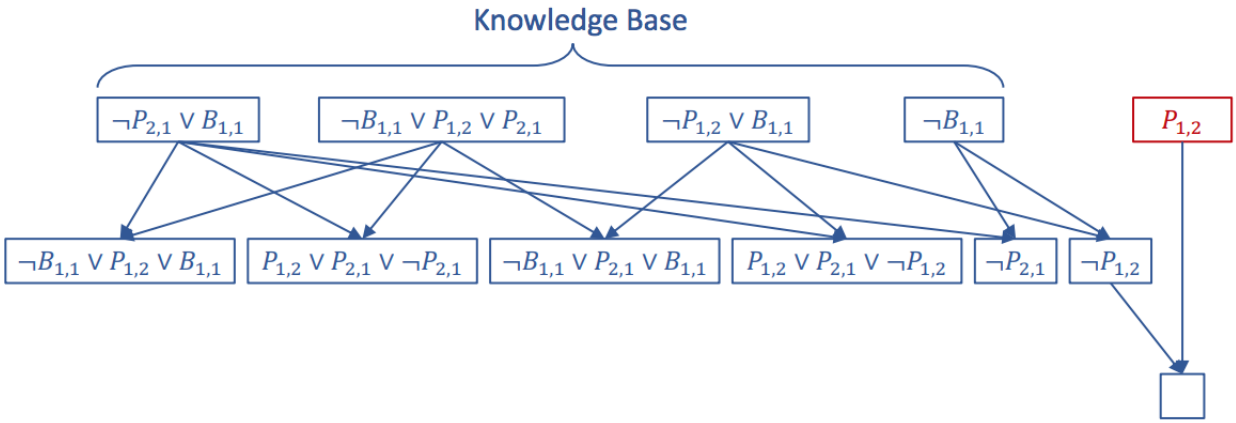
We want to prove that KB entails α

In other words, we want to prove that we cannot satisfy (KB and **not** α)

1. Start with a set of CNF clauses, including the KB as well as $\neg\alpha$
2. Keep resolving pairs of clauses until
   A. You resolve the empty clause
      Contradiction found!
      KB $\wedge \neg\alpha$ cannot be satisfied
      Return true, KB entails α
   B. No new clauses added
      Return false, KB does not entail α

From the knowledge base below, prove $\neg P_{1,2}$. Note that this is the same example as in Lecture 7. We did not have time to step through the algorithm in lecture and we'll do that now!





The resolution algorithm involves iteratively applying the general resolution rule to derive new clauses which will (hopefully) lead to a contradiction.
The general resolution rule has the following format:

$$\frac{a_1 \vee \cdots \vee a_m \vee b \qquad \neg b \vee c_1 \vee \ldots c_n}{a_1 \vee \cdots \vee a_m \vee c_1 \vee \ldots c_n}$$

We can show that this works by considering the unit clause case - resolving $(a \vee b) \wedge (\neg b \vee c)$. By the law of excluded middle, exactly one of the following cases must hold:

Case 1: b is true
In this case, c must be true for the clause $(\neg b \vee c)$ to evaluate to true.

Case 2: b is false
In this case, a must be true for the clause $(a \vee b)$ to evaluate to true.

Thus, a or c must be true for our original conjunction to be true.
We can inductively use this logic to show that the general resolution rule applies for arbitrary $m, n \in \mathbb{N}$.

A couple things to note:

- Applying the resolution rule isn't "reducing" or removing anything from our current knowledge base - rather, we're generating more clauses and adding them to our KB.
- It's not necessary to resolve all possible pairs of clauses to reach a contradiction. As we can see above, we only needed to derive $\neg P_{1,2}$ to reach a contradiction.
- We can only eliminate one pair of literals at a time. To demonstrate this, consider the following pair of clauses: $(A \vee P \vee Q) \wedge (B \vee \neg P \vee \neg Q)$. We can only resolve this pair to get $(A \vee P \vee B \vee \neg P)$ and $(A \vee Q \vee B \vee \neg Q)$ separately (which both end up evaluating to True).
  What we cannot do is derive $(A \vee B)$. Consider the model $A =$ False, $B =$ False, $P =$ True, $Q =$ False.