Recitation 7 March 1

1 Forward chaining

In this section, we will be proving a statement using forward chaining.

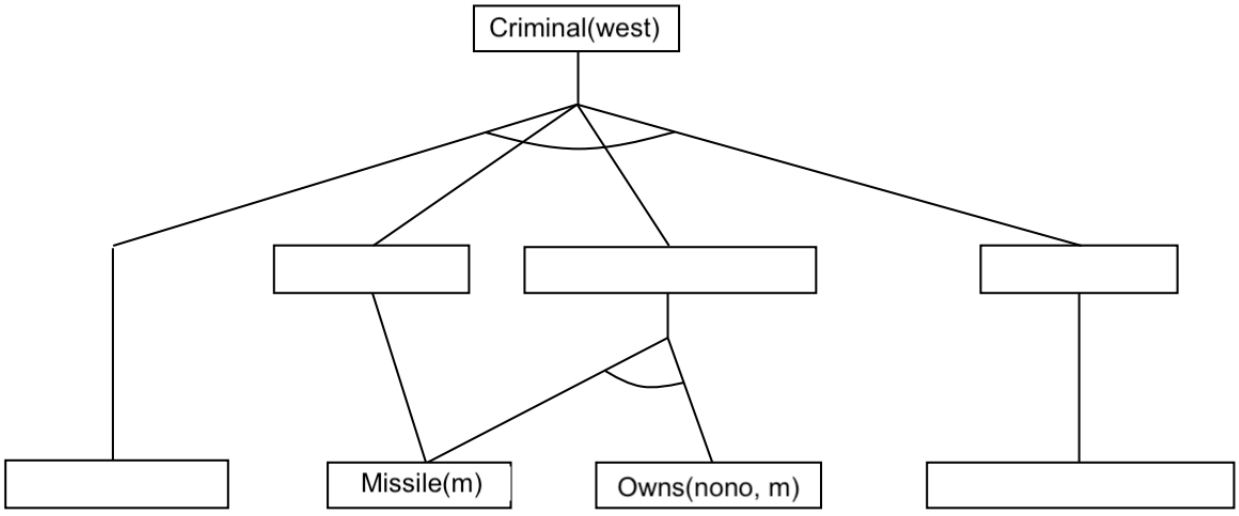
There is currently a war going on and the United States is desperate to round up all the criminals. We want to determine whether Colonel West is a criminal. Let's start with what we know.

We know that it is a crime for an American to sell weapons to hostile nations. The country Nono is an enemy of America. Furthermore, we know that Nono has some missiles, all of which were sold to it by Colonel West, who is American.

(a) Represent your knowledge base using first order logic. You can use the following function predicates: American(x), Criminal(x), Hostile(x), Missile(x), Weapon(x), Enemy(x,y), Owns(x,y), Sells(x,y,z).

1. _____ ∧ _____ ∧ _____ ∧ _____ ⇒ Criminal(x)
2. Missile(x) ⇒ _____
3. Missile(m)
4. Owns(nono, m)
5. Missile(x) ∧ _____ ⇒ Sells(west, x, nono)
6. Enemy(x, america) ⇒ _____
7. _____
8. _____

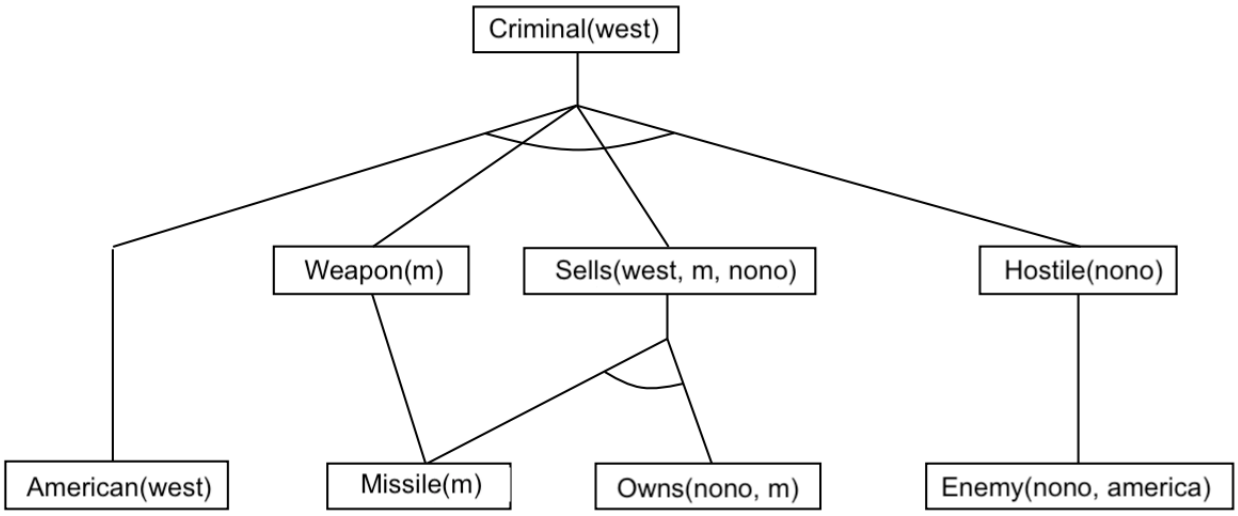
(b) Fill in the blanks below using your knowledge base to prove that Colonel West is a criminal.



(a) Represent your knowledge base using first order logic.

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Missile(x) ⇒ Weapon(x)
3. Missile(m)
4. Owns(nono, m)
5. Owns(nono, x) ∧ Missile(x) ⇒ Sells(west, x, nono)
6. Enemy(x, America) ⇒ Hostile(x)
7. American(west)
8. Enemy(nono, america)

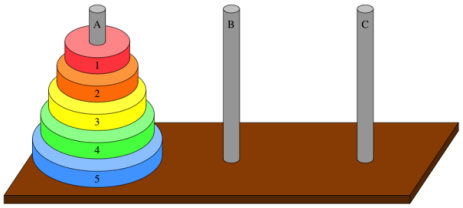
(b) Fill in the blanks below using your knowledge base to prove that Colonel West is a criminal.


Criminal(west)
American(west)
Weapon(m)
Missile(m)
Sells(west, m, nono)
Owns(nono, m)
Hostile(nono)
Enemy(nono, america)

sol.png

2 Planning Tower of Hanoi

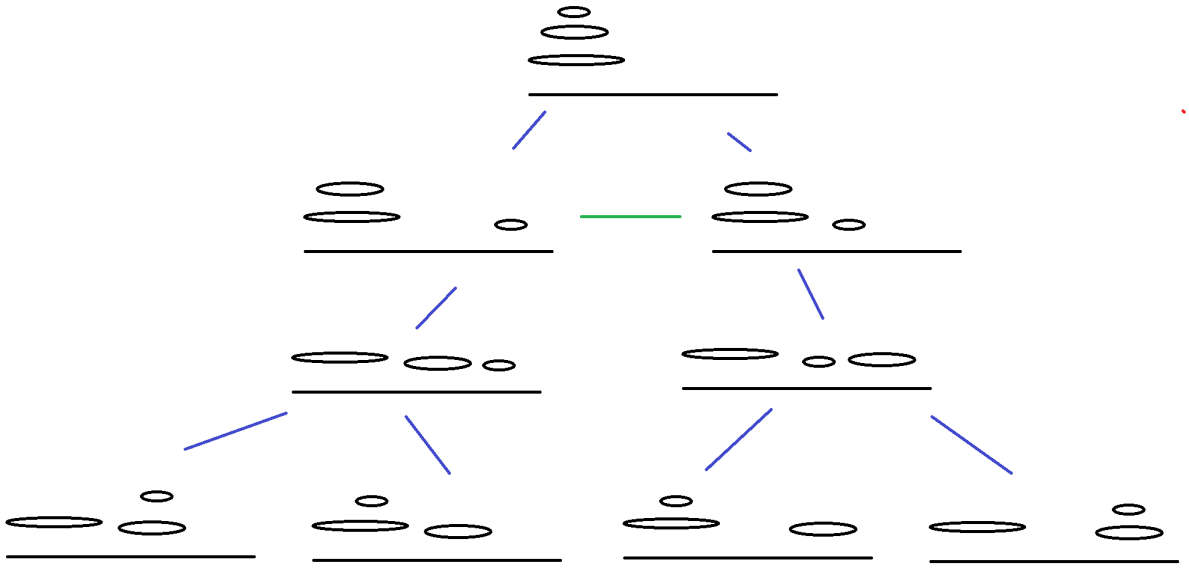
In the Tower of Hanoi problem, you are given n disks, each of a distinct size, and 3 rods, A, B and C. The disks start off stacked on top of each other on rod A, stacked from largest being the lowest to smallest being the highest in a "tower", and the goal is to move that tower to the rod C. You can only move a disk to an empty rod or on top of a larger disks, and disks may only have one other disk on its surface (they must be stacked linearly).



(a) Assume we have 3 disks. Formulate the problem as a graph-planning problem, specifying instances, operators, and start/goal states.

See provided sample code

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



(c) Generalize the problem formulation for n disks.

See provided sample code

3 Discussion-Based Questions

Let us consider forward-chaining in both a first-order logic and propositional logic setting. Find someone sitting near you to talk through the following questions with, and take some time to look through the following pseudocode snippets from the textbook.

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  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 symbols
  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
```

Figure 7.15 The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \Rightarrow Q$ and $Q \Rightarrow P$.

Figure 1: Forward-chaining algorithm for propositional logic, from p. 258 of the course textbook.

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  inputs: KB, the knowledge base, a set of first-order definite clauses
          α, the query, an atomic sentence
  local variables: new, the new sentences inferred on each iteration

  repeat until new is empty
    new ← { }
    for each rule in KB do
      (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
      for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ... ∧ p'n)
                  for some p'1, ..., p'n in KB
        q' ← SUBST(θ, q)
        if q' does not unify with some sentence already in KB or new then
          add q' to new
          φ ← UNIFY(q', α)
          if φ is not fail then return φ
    add new to KB
  return false
```

Figure 9.3 A conceptually straightforward, but very inefficient, forward-chaining algorithm. On each iteration, it adds to KB all the atomic sentences that can be inferred in one step from the implication sentences and the atomic sentences already in KB. The function STANDARDIZE-VARIABLES replaces all variables in its arguments with new ones that have not been used before.

Figure 2: Forward-chaining algorithm for first-order logic, from p. 332 of the course textbook.

(a) First things first, what are some similarities and distinctions between propositional logic and first order logic?

Propositional logic

- The world contains facts.
- Entailment in propositional logic can be computed by enumerating models
- Agents believe facts to be T/F/Unknown.

First Order logic

- The world contains facts, objects, and relations.
- Entailment in FOL is not easy. There are lots of models that need to be enumerated!
- Agents believe facts to be T/F/Unknown.

We can think of FOL as encompassing Propositional logic.

(b) Compare and contrast these two algorithms. At a high level, what similarities can you identify, and where are there differences?

The forward-chaining algorithm PL-FC-ENTAILS?(KB,q) determines if a single proposition symbol q (the query) is entailed by a knowledge base of definite clauses. It begins from known facts (positive literals) in the knowledge base. If all the premises of an implication are known, then its conclusion is added to the set of known facts. For example, if $L_{1,1}$ and Breeze are known and ($L_{1,1} \wedge$ Breeze) $\Longrightarrow$ $B_{1,1}$ is in the knowledge base, then $B_{1,1}$ can be added. This process continues until the query q is added or until no further inferences can be made.

FOL-FC-Ask(KB, α) works slightly differently. Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added. Notice that a fact is not "new" if it is just a renaming of a known fact. One sentence is a renaming of another if they are identical except for the names of the variables. For example, Likes(x, IceCream) and Likes(y, IceCream) are renamings of each other because they differ only in the choice of x or y; their meanings are identical: everyone likes ice cream.

The forward chaining algorithm for propositional logic takes in a propositional symbol while the forward chaining algorithm for FOL takes in a propositional sentence. With first order logic, you need to keep track of fewer pieces of information.

(c) Now, consider the forward-chaining algorithm presented in Figure 2. It is designed to be conceptually straightforward, but is rather inefficient. What inefficiencies can you identify in this code?

There are three possible sources of inefficiency.

(a) The "inner loop" of the algorithm involves finding all possible unifiers such that the premise of a rule unifies with a suitable set of facts in the knowledge base. This is often called pattern matching and can be very expensive.

(b) The algorithm rechecks every rule on every iteration to see whether its premises are satisfied, even if very few additions are made to the knowledge base on each iteration.

(c) The algorithm might generate many facts that are irrelevant to the goal.

Recall that this pseudocode was designed to be conceptually straightforward, and there are ways of addressing these inefficiencies!