Recitation 7 March 1

1 Forward chaining

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

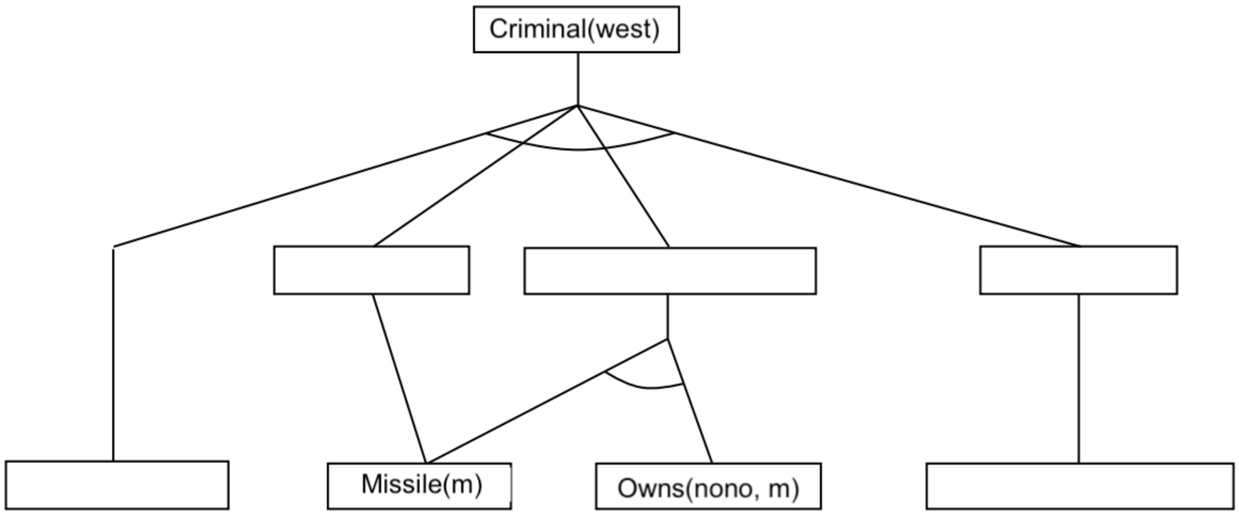
There is currently a war going on and the United States is desperate to round up all the criminals. We want to determine whether Colonel West is a criminal. Let's start with what we know.

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).

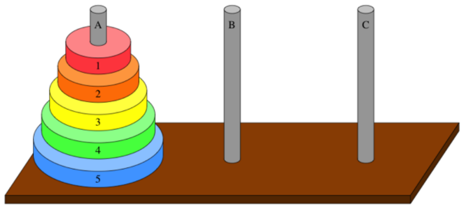
1. ______________ ∧ ______________ ∧ ______________ ∧ ______________ ⇒ Criminal(x)
2. Missile(x) ⇒ ______________
3. Missile(m)
4. Owns(nono, m)
5. Missile(x) ∧ ______________ ⇒ Sells(west, x, nono)
6. Enemy(x, america) ⇒ ______________
7. ______________
8. ______________

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



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).



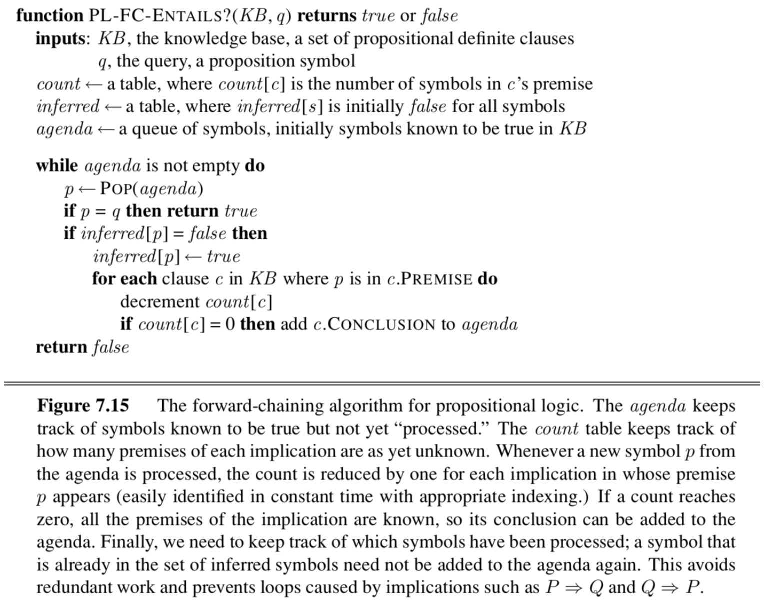
(a) Assume we have 3 disks. Formulate the problem as a graph-planning problem, specifying instances, operators, and start/goal states.

(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.

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.

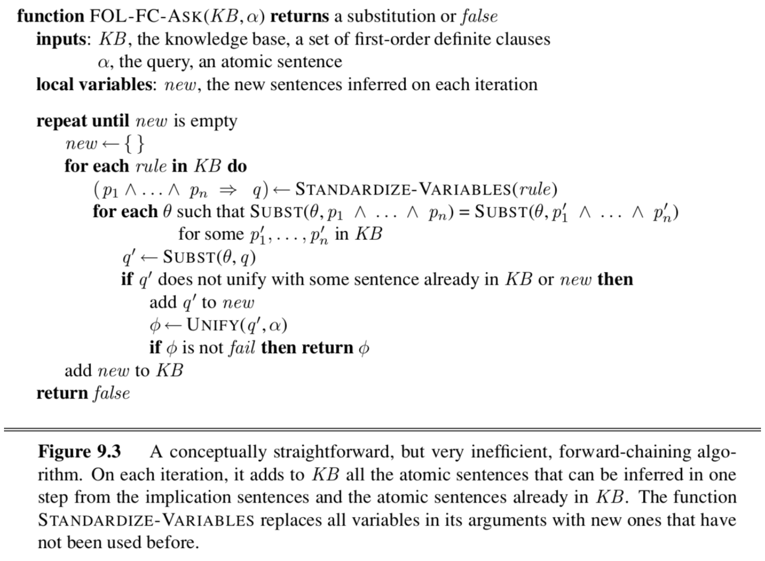


```
function PL-FC-ENTAILS?($KB, q$) returns $true$ or $false$
  inputs: $KB$, the knowledge base, a set of propositional definite clauses
          $q$, the query, a proposition symbol
  $count \leftarrow$ a table, where $count[c]$ is the number of symbols in $c$'s premise
  $inferred \leftarrow$ a table, where $inferred[s]$ is initially $false$ for all symbols
  $agenda \leftarrow$ a queue of symbols, initially symbols known to be true in $KB$

  while $agenda$ is not empty do
    $p \leftarrow$ POP($agenda$)
    if $p = q$ then return $true$
    if $inferred[p] = false$ then
      $inferred[p] \leftarrow true$
      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($KB, \alpha$) returns a substitution or $false$
  inputs: $KB$, the knowledge base, a set of first-order definite clauses
          $\alpha$, the query, an atomic sentence
  local variables: $new$, the new sentences inferred on each iteration

  repeat until $new$ is empty
    $new \leftarrow \{\}$
    for each $rule$ in $KB$ do
      $(p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES($rule$)
      for each $\theta$ such that SUBST($\theta, p_1 \land \ldots \land p_n$) = SUBST($\theta, p'_1 \land \ldots \land p'_n$)
                  for some $p'_1, \ldots, p'_n$ in $KB$
        $q' \leftarrow$ SUBST($\theta, q$)
        if $q'$ does not unify with some sentence already in $KB$ or $new$ then
          add $q'$ to $new$
          $\phi \leftarrow$ UNIFY($q', \alpha$)
          if $\phi$ is not $fail$ then return $\phi$
    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?

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

(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?