CS1571
Fall 2019
10/23 In-Class Worksheet

Name: Simran Gidwani
Where were you sitting in class today: Back right

A. Pre-Reflection

On a scale of 1-5, with 5 being most confident, how well do you think you could execute these learning objectives:

14.1 Describe definite and Horn clauses and their properties _____
14.2 Explain forward chaining _____
14.3 Analyze the implementation of forward chaining _____
14.4 Explain backward chaining _____
14.5 Analyze the implementation of backward chaining _____

B. Walk through forward chaining

1. Here is the method for propositional forward chaining from logic.py (and pseudocode from the textbook). Given the graph and proposition list, walk through the forward chaining algorithm, filling out the table as you go. You are trying to see if the KB entails Q.

```
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
```

```
def pl_fc_entails(KB, q):
        count = {c: len(conjuncts(c.args[0]))
                 for c in KB.clauses
                 if c.op == '==>'}
        inferred = defaultdict(bool)
        agenda = [s for s in KB.clauses if is_prop_symbol(s.op)]
        while agenda:
            p = agenda.pop()
            if p == q:
                return True
            if not inferred[p]:
                inferred[p] = True
                for c in KB.clauses_with_premise(p):
                    count[c] -= 1
                    if count[c] == 0:
                        agenda.append(c.args[1])
        return False
```

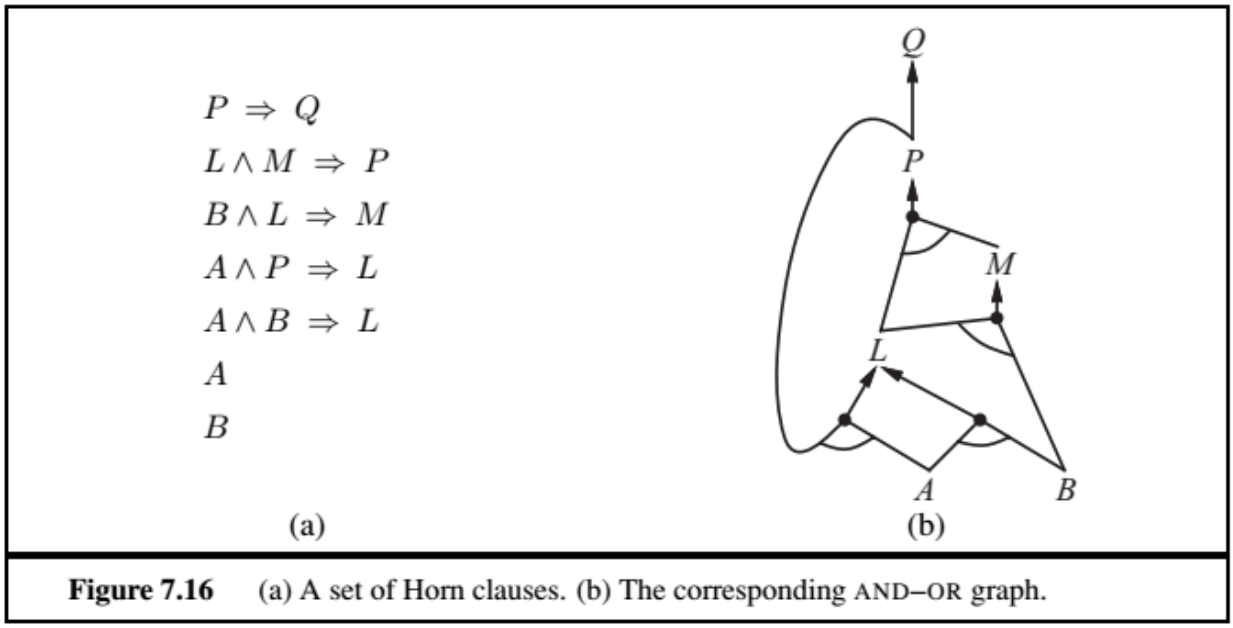

Figure 7.16 (a) A set of Horn clauses. (b) The corresponding AND–OR graph.

		Counts				
p	**Agenda**	**A^B=> L**	**A^P=> L**	**B^L=>M**	**L^M => P**	**P=>Q**
	A, B	2	2	2	2	1
A	**B**	**1**	**1**	**2**	**2**	**1**
B	**L**	**0**	**1**	**1**	**2**	**1**
L	**M**	**0**	**1**	**0**	**1**	**1**
M	**P**	**0**	**1**	**0**	**0**	**1**
P	**Q**	**0**	**0**	**0**	**0**	**0**

2. You can also do forward chaining in first-order logic. Which new facts can be inferred from this KB?

KB:

Rule 1: Steamboat (x) ^ Sailboat (y) => Faster (x, y)
Rule 2: Sailboat (y) ^ RowBoat (z) => Faster (y, z)
Rule 3: Faster (x, y) ^ Faster (y, z) => Faster (x, z)
Fact 1: Steamboat (Titanic)
Fact 2: Sailboat (Mistral)
Fact 3: Sailboat (Voyager)
Fact 4: RowBoat(PondArrow)

New Facts:

Steamboat(Titanic) ^ Sailboat(Mistral) => **Faster(Titanic, Mistral)**
Steamboat(Titanic) ^ Sailboat(Voyager) => **Faster(Titanic, Voyager)**
Sailboat(Mistral) ^ RowBoat(PondArrow) => **Faster(Mistral, PondArrow)**
Sailboat(Voyager) ^ RowBoat(PondArrow) => **Faster(Voyager, PondArrow)**
Faster(Titanic, Mistral) ^ Faster(Mistral, PondArrow) => **Faster(Titanic, PondArrow)**

C. Backward Chaining

3. Go to the FOL backward chaining code. Attempt to successfully call the backward chaining method on crime_kb to infer Criminal(West).

Once you do, trace the different calls to fol_bc_and and fol_bc_or to understand how backward chaining operates in this domain.

D. Post-Reflection

On a scale of 1-5, with 5 being most confident, how well do you think you could execute these learning objectives:

14.1	Describe definite and Horn clauses and their properties	_____
14.2	Explain forward chaining	_____
14.3	Analyze the implementation of forward chaining	_____
14.4	Explain backward chaining	_____
14.5	Analyze the implementation of backward chaining	_____