CS 5100, Fall 2011
Assignment 2—Inference and Proof in Propositional Logic

Assigned: September 15, 2011

Part I: Propositional Logic

Due: September 29, 2011 in class (hard copy)
Do the following exercises from your text:

a) Exercise 7.9 modified:

Use truth tables to verify each of the LAST 4 equivalences in FIGURE 7.11:

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

b) Exercise 7.12:

Use resolution to prove the sentence $\neg A \wedge \neg B$ from the clauses in exercise 7.20.

c) Exercise 7.13a:

Show that the clause $(\neg P_1 \vee \ldots \vee \neg P_m \vee Q)$ is logically equivalent to $(P_1 \wedge \ldots \wedge P_m) \Rightarrow Q)$.

d) Exercise 7.20 modified:

Convert the set of sentences S_1 – S_6 to clause form and identify which of the resulting clauses are Horn clauses. (NOTE: there will be more than 6 resulting clauses).

S_1: $A \Leftrightarrow (B \vee E)$.
S_2: $E \Rightarrow D$.
S_3: $C \wedge F \Rightarrow \neg B$.
S_4: $E \Rightarrow B$.
S_5: $B \Rightarrow F$.
S_6: $B \Rightarrow C$.

Part II: Implementing a Propositional Logic Reasoning System

Due: October 6, 2011 11:59pm via Blackboard

Objective: Continue to learn python and implement your first significant automated reasoning system.

Project Description: For this assignment, you will create a knowledge-based agent program that adds new percepts (atomic propositions) to a database of world knowledge, represented as a set of Horn clauses. The program will perform forward-chaining inference to add not only the new percepts but ALL the new knowledge that logically follows from combining each new percept with the agent's prior knowledge. For example, if the agent's prior knowledge includes:

$apple \Rightarrow fruit.$
$apple \Rightarrow sweet.$
$ripe.\ fruit \wedge sweet \wedge ripe \Rightarrow delicious.$

And the percept is *apple*, then *apple*, $fruit$, *sweet*, and *delicious* would all be added to the agent's knowledge base.

You may assume the knowledge base is "explicit, " i.e., it does not contain any implicit knowledge. (The question of what knowledge should be explicitly represented and what knowledge should be implicit is an important design decision in AI systems, part of the "control of inference" problem, but we will not address that problem right now).

We will use "logic programming" notation in the assignment, so the input shown above will actually look like this in your file:

fruit IF apple
ripe
sweet IF apple
delicious IF fruit sweet ripe

Implementation Notes: For full credit, you must use an object-oriented representation of the clauses in your KB, with good OO technique such as the use of `__str__` (note that both the KB items and the percepts in this assignment are definite clauses).

Input: The knowledge base will be initialized from a file `KB.txt`. The knowledge base should be loaded automatically when the source file is loaded. The order of clauses in `KB.txt` may affect the order in which answers are generated but will not change any answers.

The reasoning program will be invoked by a top-level function: `PLFC()` which starts the processing of new percepts. PLFC stands for propositional logic forward chaining.

The percepts to be processed will be in a file `percepts.txt`. Both `KB.txt` and `percepts.txt` will be in the same directory as the Python source file. You can assume the input files do not contain errors. For this assignment the input will NOT put each symbol in quotes, so therefore your program must process each line of the input as a string and not use eval() for that purpose.

Format of `KB.txt`: Prior knowledge will be represented one Horn clause per line, in "reverse" (logic programming) format:

conclusion IF $\text{premise}_1 \ldots \text{premise}_N$

This represents the logical formula: $\text{premise}_1 \wedge \text{premise}_2 \wedge \ldots \wedge \text{premise}_N \Rightarrow$ conclusion. For a known "fact", the line should just contain that proposition.

Format of `percepts.txt`: New percepts will be in the form of single propositions, one per line.

Output of `PLFC()`: Your program should print a comment confirming that the database has been initialized, along with its size (number of clauses). Each percept should be echoed when you begin to process it. Each fact added by forward chaining should be displayed, along with its immediate justifying rule from the KB. Your output should be human-readable.

Algorithm (recursive):

```
PLFC()
   # uses and modifies KB -- a global knowledge base of Horn clauses for each new
   # percept p
   PLFC1(p) #use a recursive "helper function"


   PLFC1(percept)
      if percept is already known, return
      else
         add percept to knowledge base
         for r in rules where conclusion of r is not already known
            if percept is a premise of r and all the other premises of r are known
               PLFC1(conclusion of r)
```

Example `KB.txt` and `percepts.txt` files will be provided with the program description, representing knowledge about healthy and unhealthy food. The example files will NOT include every situation your program needs to handle, so you are responsible for creating additional test cases.

Extra Credit: Extend your program to handle the situation where `KB.txt` might contain implicit knowledge. For example, the KB could be:

b IF a
c IF b
r IF c and d
a

This KB implicitly contains the knowledge that b and c are true. If the percept d is found in the input, the PLFC algorithm above will NOT be able to infer r even though logically it follows from d + KB. One solution is to fully expand the knowledge in KB before you start processing the percepts. This would mean a new KB:

b IF a
c IF b
r IF c and d
a
b
c

Using this "explicit" KB, if the percept is d, the algorithm above would be able to infer r. **NOTE:** It is highly recommended to get the required version of `PLFC()` working correctly before you attempt to extend it to handle the case of an incomplete KB.

Submission: Upload your COMMENTED Python source file to Blackboard. Please give the source file a name that includes your last name (or part of it) and proj2, such as: sliva_proj2.py