CS 5100, Fall 2011
Assignment 3—First-order Logic

Assigned: October 6, 2011

Part I: First-order Logic

Due: October 13, 2011 in class (hard copy)
Do the following exercises from your textbook:

a) Exercise 9.4.

b) Exercise 9.6.

Identify the logial language elements you use for this problem: constants, function symbols (with arity), predicate symbols (with arity).

c) Convert the 6 kinship axioms from section 8.3 of your textbook to KB in Conjunctive Normal Form (CNF). Follow the procedure in the text and in teh slides shown in lecture.

d) Draw the AND/OR tree representing backward chaining for the knowledge base shown below for each of the two queries, AND SHOW WHAT ANSWER WOULD BE RETURNED by a backward chaining inference engine (show all answers if there are more than one). The format of the AND/OR tree is the extended format defined in class.

Queries:

i. $Plays(?x, Basketball)$
ii. $Is - happy(?x)$

Knowledge base:

R1. $Likes(x, Basketball) \Rightarrow Plays(x, Basketball)$
R2. $Plays(x, Basketball) \Rightarrow Is - healthy(x)$
R3. $Has - pets(x) \Rightarrow Is - happy(x)$
R4. $Is - healthy(x) \wedge Is - rich(x) \Rightarrow Is - happy(x)$
F1. $Likes(John, Basketball)$
F2. $Plays(Sam, Basketball)$
F3. $Is - rich(Sam)$
F4. $Has - pets(Sally)$

Part II: Implementing a Backward Chaining Program in Python

Due: October 20, 2011 11:59pm via Blackboard

Objective: Implement an automated reasoning system in first-order logic using backward chaining.

Project Description: For this assignment, you will create a backward chaining program in Python. Your program will have the top level function `FOLBC(goal`, which returns a list of substitutions that will make the goal true. `goal` is an instance of a single `FOLExp` (the provided class for representing first-order logic expressions). `FOLBC(goal)` should include a recursive helper function `FOLBC1(goal,subst)`. See the example below:

```
function FOLBC(goal) # goal is a list of FOLexp

   # ans is a list of substitutions that make goal true, goal is a list of FOLexp,
   # and θ =[ ] are the substitutions so far (initially empty)
   ans = FOLBC1([goal], [ ] )

   return cleanup(ans) #only include variables mentioned in query
```

Your program must also include a top level function called `FOLBCtest()` of NO arguments that gets queries from a file `queries.txt` and one by one:

1. prints the query
2. calls `FOLBC` with the query (represented as an `FOLexp`) as its argument
3. prints the result in human readable form

```
function FOLBCtest()

   for each query in queries.txt
      print query
      ans = FOLBC([FOLexp(query)])

   print answers from ans in human readable form
```

Input: Your program (as in the forward chaining case), will use a global knowledge base KB, which you will be read from a file `KB.txt` consisting of definite clauses:

Atomic-formula
Atomic-formula IF Atomic-formula AND Atomic-Formula . . .

Example:
Likes[John, Pizza]

Likes[x, Pizza] IF Likes[x, Tomato] AND Likes[x,Cheese]
Likes[x, Pizza] IF Likes[Father[x], Pizza]

Variables begin with lower case letters.
Constants, predicate symbols, and function symbols begin with upper case letters.
Arguments are represented using Python's list format.

A query will consist of a list of `FOLexp`. There will be one query per line in the `queries.txt` input file. To get full credit you should find and print ALL the correct answers to the query. If there are no variables in the query then the answer will just be: Yes or No indicating that it was proved or not proved. The readability of your output will be part of your grade. Remember, no changes are made to the KB so each new query is independent of the previous queries.

Some simple test files have been provided, but you should not depend on them to handle every case.

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_proj3.py