

Cooperating Intelligent Systems Lab 3

Logic

July 6, 2009

1 Task

The main purpose of this assignment is to acquaint the student with the practical/computational aspects of logical reasoning. Some insights of how expertize can be used to develop knowledge-based systems are likely to result. Moreover, it is expected that the student will understand one way (a standard way) of how declarative knowledge can be incorporated and used in a procedural/imperative program. If you have read the material on logic programming above you should have a fair understanding of the basic steps of a so-called theorem-prover. The assignment builds on a documented attempt to build a theorem-prover in Java.

2 Details

In the distributed code (ailogic.zip) I have included a simple example, described below in Horn clauses:

1. $\text{uncle}(\text{Uncle}, \text{Nephew}) < - \text{brother}(\text{Uncle}, X), \text{child}(\text{Nephew}, X).$
2. $\text{brother}(X, Y) < - \text{brother}(Y, X).$
3. $\text{brother}(\text{douglas}, \text{danny}).$
4. $\text{child}(X, Y) < - \text{son}(X, Y)$
5. $\text{child}(X, Y) < - \text{daughter}(X, Y)$

6. son(steven, danny).
7. daughter(kelly, douglas).

The goal of the example is to infer who kellys uncle is. $\leftarrow \text{uncle}(X, \text{kelly})$.
Now, carry out the following steps.

- Step 1; Run the example program to check the results. Try to understand what the program does. Provide a brief explanation (3-5 sentences) of each of the Java classes. Run the code by typing `javac jlogic/Example.java` and `java jlogic.Example`.
- Step 2; Rewrite the predicates so that it is possible to express knowledge of gender. Add predicates for mother, father, sister, aunt, grandmother/grandfather. Add more people to the family tree and check if everything works. Submit your program and the final substitutions generated by the program for at least three interesting queries.
- Step 3; Write a new example application using the blocks domain. Add the rules
 - 'above(X, Y)' true if X is on top of Y.
 - 'under(X)' true if X has blocks above itself.
 - 'ontop(X)' true if X has no blocks above itself.

Assume that you have at least two stacks of blocks.

- Step 4; Write a function 'move' that adds and removes facts about blocks, i.e. a function that asserts and retracts on-facts, e.g. if block a is moved from b to c you would retract 'on(a, b)' and assert 'on(a, c)'. Importantly, you need to check so that 'ontop(a)' and 'ontop(c)' are true first. You should ofcourse make use of the logical inference available to you.

3 Grading

Your results are graded as either u,3,4 or 5. These are the requirements for the grades:

3. Completed step 1 and 2
4. Same as for grade 3 plus step 3.
5. Same as for grade 4 plus step 4.