Assignment 2
CSE 473 Autumn 2010

October 22, 2010

The assignment is graded out of 100 points and is due November 5 by email to `guillory@cs.washington.edu` before class. Attach to this email a zip file containing your code and answers to the questions. Put your answers in a text file titled "answers.txt". Be sure to number your answers.

Create a Knowledge Base (40 points)

Problem 1. (21 points) Using the following predicates

- $ParentOf(A, B)$ (A is a parent of B)
- $AncestorOf(A, B)$ (A is an ancestor of B)
- $Siblings(A, B)$ (A is a sibling to B)
- $Cousins(A, B)$ (A is a cousin to B)
- $BloodRelated(A, B)$ (A is blood related to B)

Convert the following statements into first order logic. Write your answers in plain text in "answers.txt" using `&`, `|`, `->`, `forall`, and `!` in place of $\vee$, $\wedge$, $\Rightarrow$, $\forall$, and $\neg$.
A. (3 points) Someone's parent is also someone's ancestor
B. (3 points) If a first person is a sibling to a second person, then the second person is also a sibling to the first person.
C. (3 points) The ancestors of someone's ancestors are also that someone's ancestors.
D. (3 points) If two people's parents are siblings, then they are cousins
E. (3 points) If a first person is a blood related to a second person, then the second person is also blood related to the first person.
F. (3 points) A person is blood related to their ancestors
G. (3 points) If a person is blood related to someone, then they are also blood related to people for whom that someone is an ancestor.

Problem 2. (13 points) Convert your knowledge base into conjunctive normal form and then convert it into propositional logic using the following set of 5 ground terms: `Frank`, `Bob`, `Jill`, `Mary`, `Tom`. In addition to the rules above, add to your knoweldge base the following assertions:

- $ParentOf(Jill, Frank)$
- $ParentOf(Bob, Mary)$
- $ParentOf(Tom, Bob)$
- $ParentOf(Tom, Jill)$
- $Siblings(Jill, Bob)$

Save the result into a text file "knowledgebase.txt" using the following format:

- Each line of the text file is a clause
- A clause is written as a sequence of literals each separated by a space
- Each literal is written as a proposition which may or may not be preceeded by a "!" indicating negation
- A proposition is written as a string of characters which does not contain any spaces

For example, the knowledge base $BloodRelated(Frank, Jill) \wedge (BloodRelated(Tom, Jan) \vee \neg Siblings(Tom, Jan))$ would be written as

```
BloodRelated(Frank,Jill)
BloodRelated(Tom,Jan) !Siblings(Tom,Jan)
```

Note that your final knowledge base will have close to 1000 clauses, so you probably don't want to do this by hand. To help you out, we've provided a small amount of Java code. You can download this code from `http://www.cs.washington.edu/education/courses/cse473/10au/a2.zip`. This code provides basic data structures for reprsenting a knowledge base in CNF form and code for reading and writing the format described above. We've also provided a function which expands a knowledge base by replacing variables with ground terms. See the function named "propositionalize" in the file "Main.java" and read the code and comments there to see how it works. You do not have to use this code, but you will probably find it helpful.

Problem 3. (6 points) What about this knowledge base makes it easy to propositionalize? What about this knowledge base makes it easy to run inference on?

Implement and test WalkSAT (60 points)

Problem 4. (30 points) Implement WalkSAT as it is writen in the book. Your implementation should take in a knowledge base in the format described above and two parameters p and $maxFlips$. It should then output $true$ if the knowledge base is satisfiable and $false$ otherwise.

We suggest you use the code we've provided as a starting point: `http://www.cs.washington.edu/education/courses/cse473/10au/a2.zip`. We don't require you use this code however. If you use a language other than Java, please contact the TA to be sure that we will be able to read and run your code.

Problem 5. (10 points) WalkSAT, like any satisfiability solver, can be used for inference. How can we do this? WalkSAT sometimes returns $false$ even if the knowledge base is in fact satisfiable. What sort of errors will WalkSAT sometimes make when we use it for inference? Is it sound? Complete?

Problem 6. (10 points) Test your WalkSAT implementation by using it to answer the following inference queries:

- $Cousins(Frank, Mary)$
- $AncestorOf(Tom, Mary)$
- $BloodRelated(Tom, Frank)$
- $AncestorOf(Bob, Frank)$
- $Cousins(Jill, Bob)$

Use parameters $p = .5$ and $maxFlips = 10000$ and report back the results. Run each inference multiple times. Does your implementation ever report back the wrong answer on these queries using these parameters?

Problem 7. (10 points) Repeat the experiments above using different values of p and $maxFlips$ and report the results. What happens if you make $maxFlips$ smaller? Larger? How sensitive is the algorithm to changes in p?

Problem 8. Extra Credit (10 points) There are two major performance bottlenecks in the WalkSAT algorithm: (1) computing the set of unsatisfied clauses and (2) computing the change in the number of unsatisfied clauses as a result of flipping a truth value. High performance implementations of WalkSAT speed up these computations through clever data structures.

Modify your implementation of WalkSAT to speed up one or both of these computations. Describe the changes you made. How much faster is your new implementation in theory? In practice?