CSCI 561 - Foundation of Artificial Intelligence

Programming Assignment 2: Logical Inference
Due April 24, 2013 before the class
Submit via Blackboard

Objective

In this assignment, you will practice inference in propositional and first-order logics. The programming part of the assignment (Tasks 1-3) deals with propositional inference and the written question focuses on inference in first-order logic.

The aim of logical inference is to decide whether $KB \models \alpha$ for some sentence α. You will implement the resolution algorithm to determine the entailment. To show that $KB \models \alpha$, we show that $(KB \wedge \neg\alpha)$ is unsatisfiable. First $(KB \wedge \neg\alpha)$ is converted into Conjunctive Normal Form (CNF) and the resolution rule is applied to the set of the resulting clauses. the resolution inference rule for CNF is:

$$\frac{l_1 \vee \cdots \vee l_k \quad m_1 \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where l_i and m_j are complementary elements. Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present. The resolution rule applied to a contradiction $P \wedge \neg P$ yields the empty clause. If an empty clause is derived then $KB \models \alpha$. If there are no new clauses that can be added and an empty clause has not been derived, KB does not entail α.

Input Format

The program will take in as input two files. One of them will store the knowledge base and the other one will contain some statements whose entailment you need to determine. Both files will contain one or more logical statements (one per line). Each logical statement may consist of literals, logical symbols, and parentheses. Literals are strings that consist only of (lower-case and/or upper-case) letters, and the allowed logical symbols are `AND`, `OR`, `NOT`, `=>`, and `<=>`. There is no limit on the maximum level of nested statements, i.e., the number of parentheses within parentheses. Files are stored in prefix format for which it is easier to write parsing code. An example of a logical statement is `(OR (AND A B) (AND`

`C (=> (NOT D) E)))`[1]. For simplicity, assume that each `AND/OR` operator can only have two operands.[2] The knowledge base should be considered to be a conjunction of all the statements in the corresponding file.

Task 1: 30 points

Your program will convert the knowledge base into CNF. Assuming that you are using C/C++, and your executable is called "program", this is what the command line should look like for calling the program[3]:

```
program -task1 [kb_filename] [output_filename]
```

For example:

```
program -task1 kb.txt output.txt
```

Each line of the output will contain exactly one clause of the resulting CNF printed in prefix format. This means that each line will only contain the `OR` and `NOT` logic symbols and the knowledge base will be a conjunction of all the lines. Your program should provide the answer in less than 1 minute. Note that it is not necessary to generate the most compact CNF form.

Task 2: 20 points

Your code will identify whether the knowledge base is valid, satisfiable, or unsatisfiable. The program will be called as below:

```
program -task2 [kb_filename] [output_filename]
```

For example:

```
program -task2 kb.txt output.txt
```

The program should print out "UNSATISFIABLE" if the knowledge base is unsatisfiable, "VALID" if the knowledge base is valid, and "SATISFIABLE" otherwise. If a decision cannot be made in 5 minutes, you should print out "NOT TERMINATED".

[1] The equivalent infix format is: `(A AND B) OR (C AND (NOT D => E))`

[2] For example, instead of the sentence `(AND A B C)`, we will have `(AND A (AND B C))`.

[3] In Java, it looks like `java program -task1 [kb_filename] [output_filename]` where `program` is your main java class.

Task 3: 30 points

For each logical sentence in the statements file, your code will determine whether the knowledge base entails the sentence or not. The program should be invoked as follows:

```
program -task3 [kb_filename] [statements_filename] [output_filename]
```

For example:

```
program -task3 kb.txt statements.txt output.txt
```

The program will generate one output line in the output file for each sentence of the statements file: "ENTAILED" if the knowledge base entails the sentence, "NOT ENTAILED" otherwise. For each sentence, if your program cannot come up with the answer in 5 minutes, it should print out "NOT TERMINATED".

Example for Tasks 1-3

Suppose that we have the following knowledge base and statements files:

kb.txt

```
(=> W H)
(=> C H)
(NOT H)
(OR (AND A B) (AND A C))
(<=> A D)
```

statements.txt

```
(AND (NOT C) (NOT W))
(=> B H)
(A)
```

Now, we show the expected results of running the program with different parameters.

Task 1: *program -task*1 *kb.txt output.txt*

output.txt

```
(OR (NOT W) H)
(OR (NOT C) H)
(NOT H)
(A)
```

```
(OR B C)
(OR (NOT A) D)
(OR (NOT D) A)
```

Task 2: *program -task2 kb.txt output.txt*

output.txt

```
SATISFIABLE
```

Task 3: *program -task3 kb.txt statements.txt output.txt*

output.txt

```
ENTAILED
NOT ENTAILED
ENTAILED
```

Coding Requirements

Your program must be written in C/C++ or Java. For C/C++, you may use the Standard Template Library (STL) and for Java, you are allowed to use Java (version 1.6 or lower versions) standard libraries. You may not use any 3rd party objects/algorithms without express permission from the course TA or Professor. Your program must compile and run on Aludra (see: `http://www.usc.edu/its/web/getting_started/ppages.html` if you are unfamiliar with Aludra). You will be provided with two knowledge bases along with the outputs of running different tasks on them. When grading the assignment, we will test your program with three more knowledge bases in addition to the provided ones. The size of the test knowledge bases is in the oder of the size of the given test cases. The programs that do not terminate in the time frame specified for a task or part of a task will not get the credit for that part.

Written Question: 20 points

You need to solve this question by hand. Consider the following first-order logic knowledge base:

$IsProf(Alice)$
$\forall x(IsProf(x) \rightarrow IsPerson(x))$
$IsDean(John)$
$\forall x(IsDean(x) \rightarrow IsProf(x))$
$\forall x(\forall y(IsProf(x) \wedge IsDean(y) \rightarrow IsFriendOf(y, x) \vee \neg knows(x, y)))$

$\forall x(\exists y(IsFriendOf(y,x)))$
$\forall x(\forall y(IsPerson(x) \wedge IsPerson(y) \wedge Criticize(x,y) \rightarrow \neg IsFriendOf(y,x)))$
$Criticize(Alice, John)$

Using both Resolution and Backward Chaining show that $\neg IsFriend(John, Alice)$ can be inferred from the knowledge base. For backward chaining, draw a diagram like Fig. 9.7 of the textbook (page 338) and for resolution, after converting to CNF, draw a diagram like what is shown in Fig. 9.11 (page 348). In both diagrams, you must show the bindings of the unifications. If it is easier for you, you can handwrite your answer, scan it or take a photo of it, and then convert it to pdf.

Submission

You must submit the following files/folders in a single .zip archive named `Firstname_Lastname_project2.zip` and submit it via Blackboard:

- **README.txt**: This file provides required instructions to run your program. It must contain your full name, USC login name and USC ID and any additional information you want us to know when grading your work.
- **Firstname_Lastname_project2_report.pdf**: A pdf containing your answer to the written question.
- **Source**: This folder has all .c/.cpp/.h/.java files required to compile and run your code. You should have a Makefile in this folder that compiles your source code.

Ground Rules

No collaboration will be allowed on this project. It must reflect just the work of the individual student, with no outside help (except for questions asked of the instructor or TA). Use of any code not produced by the individual student whether from a friend, the Internet, or anywhere else without explicit permission from the instructor or TA is explicitly forbidden. **The standard penalty for violating this policy is an F in the course.**

You are allowed a total of two late days that can be used on the programming projects. This means that, without a penalty, project one can be two days late OR project two can be two days late OR each project can be one day late. Once you have used up your two late days, one additional day late will result in a 25% reduction in the total score, two additional days late will yield a 50% reduction, and no credit will be given for three or more additional days late. Late days are in units of days, not hours, so using up part of a day uses up the whole day.