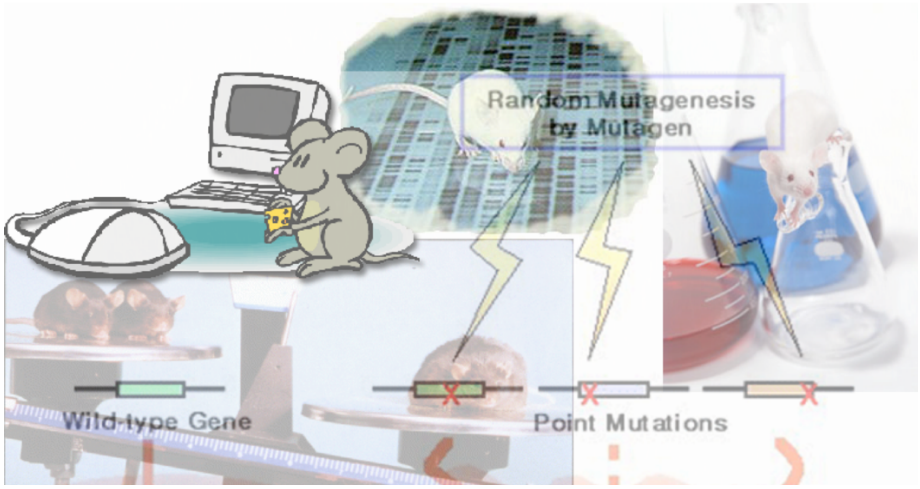
**CSCI-561 - Fall 2016 - Foundations of Artificial Intelligence**
**Homework 3**

**Due November 21, 2016 23:59:59**




**Guidelines**

This is a programming assignment. You will be provided sample inputs and outputs (see below). Please understand that the goal of the samples is to check that you can correctly parse the problem definitions, and generate a correctly formatted output. The samples are very simple and it should not be assumed that if your program works on the samples it will work on all test cases. There will be more complex test cases and it is your task to make sure that your program will work correctly on any valid input. You are encouraged to try your own test cases to check how your program would behave in some complex special case that you might think of. Since **each homework is checked via an automated A.I. script**, your output should match the example format *exactly*. Failure to do so will most certainly cost some points. The output format is simple and examples are provided. You should upload and test your code on vocareum.com, and you will submit it there. You may use any of the programming languages provided by vocareum.com.

**Grading**

Your code will be tested as follows: Your program should take no command-line arguments. It should read a text file called "input.txt" in the current directory that contains a problem definition. It should write a file "output.txt" with your solution. Format for files input.txt and output.txt is specified below. End-of-line convention is Unix (since vocareum is a Unix system).

The grading A.I. script will, 50 times:

- Create an input.txt file, delete any old output.txt file.
- Run your code.
- Compare output.txt created by your program with the correct one.
- If your outputs for all 50 test cases are correct, you get 100 points.
- If one or more test case fails, you get 50 – N points where N is the number of failed test cases.

Note that if your code does not compile, or somehow fails to load and parse input.txt, or writes an incorrectly formatted output.txt, or no output.txt at all, or OuTpUt.TxT, **you will get zero points**. Please test your program with the provided sample files to avoid this. You can submit code as many times as you wish on vocareum, and the last submitted version will be used for grading.

**Academic Honesty and Integrity**

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe you are urged to err on the side of caution. Do not copy work from another student or off the web. Sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

**Do not copy** code or written material from another student. Even single lines of code should not be copied.

**Do not collaborate** on this assignment. The assignment is to be solved individually.

**Do not copy** code off the web. This is easier to detect than you may think.

**Do not share** any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones considered below.

**Do not copy** code from past students. We keep copies of past work to check for this.

**Do** ask the professor or TA if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

**Project description**

Dr. Ernest Beakerman has been experimenting with gene knockout mice for the last 10 years. Recently he has had a major breakthrough with his KM-52a strain. It turns out that the mice are hyper intelligent and possess a keen grasp of logic far superior to any human. Since mice by their nature are quite friendly the KM-52a mice have employed themselves to bettering humanity. For instance, Algernon II is now advising the president on foreign matters and has brought the ISIS crisis to a graceful ending and stopped the spread of AIDS in Africa.

The problem with KM-52a is that, like all other mice, they live at most 4 years. As a result, new generations must be constantly trained to replace the current mice, which die off very quickly. Additionally, they must be trained with great speed so that they can spend as much time as possible aiding humanity into a new era of prosperity.

Dr. Beakerman has employed you in his lab to facilitate the training of KM-52a mice. Each mouse will sit in front of a computer practicing logic exercises. It will be presented with several first-order logic statements. It will read them and then type in a logical query, which the statements should be able to prove. Your job is to write a program that will check each conclusion a mouse makes and give it immediate feedback as to whether the query can be proven. Thus, the mice will learn very quickly via feedback a strong understanding of logic.

To help you develop your system, Dr Beakerman has provided you with several sample knowledge bases the mice have produced. The knowledge bases contain sentences with the following defined operators:

| | |
|---|---|
| `NOT X` | `~X` |
| `X OR Y` | `X \| Y` |
| `X AND Y` | `X & Y` |
| `X IMPLIES Y` | `X => Y` |

**Your assignment is:**

You will use **the resolution inference algorithm, full first-order version** (see slide 76 of Monday-Wednesday slides for session14-15) to solve this problem.

Once you have completed this project Dr. Beakerman will use it to expedite the training of KM-52a mice and thus usher in a new era of peace and prosperity. After that he will collect his Nobel prize and from then on forget to ever mention your name when talking about how to train KM-52a mice.

**Format for input.txt:**

```
<NQ = NUMBER OF QUERIES>
<QUERY 1>
…
<QUERY NQ>
<NS = NUMBER OF GIVEN SENTENCES IN THE KNOWLEDGE BASE>
<SENTENCE 1>
…
<SENTENCE NS>
```

where

- Each query will be a single literal of the form Predicate(Constant) or ~Predicate(Constant).
- Variables are all single lowercase letters.
- All predicates (such as Sibling) and constants (such as John) are case-sensitive alphabetical strings that begin with an uppercase letter.
- Each predicate takes at least one argument. Predicates will take at most 100 arguments. A given predicate name will not appear with different number of arguments.
- There will be at most 100 queries and 1000 sentences in the knowledge base.
- See the sample input below for spacing patterns.
- You can assume that the input format is exactly as it is described. There will be no syntax errors in the given input.

**Format for output.txt:**

For each query, determine if that query can be inferred from the knowledge base or not, one query per line:

```
<ANSWER 1>
…
<ANSWER NQ>
```

where

each answer should be either TRUE if you can prove that the corresponding query sentence is true given the knowledge base, or FALSE if you cannot.

**Notes and hints:**

- Please name your program "**homework.xxx**" where 'xxx' is the extension for the programming language you choose. ("**py**" for python, "**cpp**" for C++, and "**java**" for Java). If you are using C++11, then the name of your file should be "**homework11.cpp**" and if you are using python3.4 then the name of your file should be "**homework3.py**".
- If you decide that the given statement can be inferred from the knowledge base, every variable in each sentence used in the proving process should be unified with a Constant. So, if you have something like A(x) => B(John), and you cannot find any x to fulfill the A(x) premise, you cannot say that B(John) is true.
- All variables are assumed to be universally quantified. There is no existential quantifier in this homework. There is no need for Skolem functions or Skolem constants.
- Operator priorities apply. Parentheses may be used in the sentences given to you for the KB, to group terms, e.g., (A(x) | B(x)) => C(x). There will not be any empty parentheses: ().
- It is recommended that you convert the KB to CNF as a pre-processing step. This means using the definition of implication to eliminate it, working negations into parentheses, possibly using distributivity rules to simplify some parentheses, and possibly splitting sentences that contain AND operators into several sentences.
- The knowledge base that you will be given is consistent. So there are no contracting rules or facts in the knowledge base.
- If you run into a loop and there is no alternative path you can try, report FALSE. An example for this would be having two rules 1) A(x) => B(x) 2) B(x) => A(x) and wanting to prove A(John). In this case your program should report FALSE.
- Considering that the size of knowledge base is not small, we recommend using an indexing method for storing your knowledge base. Such as table-based indexing or tree-based index that you have learned in class.
- You are free to use any parsing tool such as LEX and YACC if you want (As long as it runs on Vocareum).

**Example 1:**

For this input.txt:

```
6
F(Bob)
H(John)
~H(Alice)
~H(John)
G(Bob)
G(Alice)
14
```

```
A(x) => H(x)
D(x,y) => ~H(y)
B(x,y) & C(x,y) => A(x)                    [beware of operator priority]
B(John,Alice)
B(John,Bob)
(D(x,y) & F(y)) => C(x,y)                  [note parentheses on premises: not
D(John,Alice)                                    strictly required but legal]
F(Bob)
D(John,Bob)
F(x) => G(x)
G(x) => H(x)
H(x) => F(x)
R(x) => H(x)
R(Alice)
```

your output.txt should be:

```
TRUE
TRUE
TRUE
FALSE
FALSE
TRUE
```

**Example 2:**

For this input.txt:

```
2
Ancestor(Liz,Billy)
Ancestor(Liz,Bob)
6
Mother(Liz,Charley)
Father(Charley,Billy)
~Mother(x,y) | Parent(x,y)
~Father(x,y) | Parent(x,y)
~Parent(x,y) | Ancestor(x,y)
~(Parent(x,y) & Ancestor(y,z)) | Ancestor(x,z)
```

your output.txt should be:

```
TRUE
FALSE
```

**Example 3:**

Try for yourself the example from slide 92 of session14-15 of Monday-Wednesday lectures.