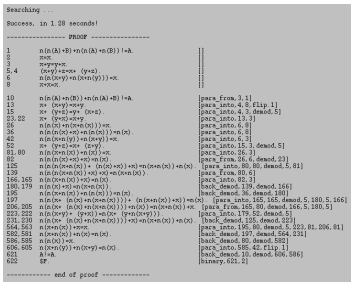
CSCI-561 - Fall 2017 - Foundations of Artificial Intelligence
Homework 3

Due November 20, 2017 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 **lose 2 points for each failed test case**. (note that one test case involves several answers in this HW; if any answer on a given test case is wrong, then the whole case is counted as wrong).

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

After the high-profile lawsuit in which you succeeded to bring RealityMan to justice after proving that distributing his Nintendo emulator was a criminal act, everyone wants to hire you! From disputes over tech patents, to lawsuits on questions of privacy in social media, to suits on liability issues with self-driving cars, to disputes between Hollywood celebrities and their agents or producers, you are just running out of time and energy to run all these cases by hand like we have done in the lectures.

Because of the highly sensitive nature of the cases you handle, and because of the extremely high monetary amounts involved, you cannot trust any existing. You thus decide to create your own, ultra-optimized inference engine.

After much debating, you decide that the knowledge bases which you will create to handle each case will contain sentences with the following defined operators:

```
NOT X                ~X
X OR Y               X | Y
```

Your assignment is:

You will use **first-order logic resolution** to solve this problem.

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 (i.e., unify variables to constants before you trigger a step of resolution).

- 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 (negation has higher priority than disjunction). There will be no parentheses in the sentences, other than around arguments of predicates.
- 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) and ***(2)*** ~B(x) | A(x) and wanting to prove A(John). In this case your program should report FALSE.

Example 1:

For this input.txt:

```
6
F(Joe)
H(John)
~H(Alice)
~H(John)
G(Joe)
G(Tom)
14
~F(x) | G(x)
~G(x) | H(x)
~H(x) | F(x)
~R(x) | H(x)
~A(x) | H(x)
~D(x,y) | ~H(y)
~B(x,y) | ~C(x,y) | A(x)
B(John,Alice)
B(John,Joe)
~D(x,y) | ~Q(y) | C(x,y)
D(John,Alice)
Q(Joe)
D(John,Joe)
R(Tom)
```

your output.txt should be:

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

Example 2:

For this input.txt:

```
2
Ancestor(Liz,Billy)
Ancestor(Liz,Joe)
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:

For this input.txt:

```
1
Criminal(West)
6
~American(x) | ~Weapon(y) | ~Sells(x,y,z) | ~Enemy(z,America) | Criminal(x)
Owns(Nono,M1)
Missile(M1)
~Missile(x) | ~Owns(Nono,x) | Sells(West,x,Nono)
~Missile(x) | Weapon(x)
Enemy(Nono,America)
```

your output.txt should be:

```
TRUE
```