Assignment 7: Theorem Prover
CS 730/830, Fall 2015
Electronic submission due at **11:30pm on Wed, Oct 21**
Hardcopy submission due at **start of class on Thu, Oct 22**

Overview

You will extend assignment 6 into a resolution refutation theorem prover for first-order logic (without equality). Given a first-order theory and query, your program will either report a series of proof steps, report that the query is false, or proceed to derive new conclusions forever. Pages 255 and 347 of the textbook might be helpful. You should implement the 'set of support' strategy (p. 355 in the textbook).

Input

As before, you may assume that all input will be in Skolemized CNF. Standard input will contain the theory (one clause per line), followed by the line `--- negated query ---`, followed by the query (on one or more additional lines). The query will already be negated. You do not need to handle equality specially. The goal of your prover is to find a contradiction between the theory and the negated query by deriving the empty clause. This proves the query.

Example input:

```
-Human(x1) | Mortal(x1)
Human(Socrates)
Animal(F2(x2)) | Loves(F1(x2), x2)
-Loves(x2, F2(x2)) | Loves(F1(x2), x2)
--- negated query ---
-Mortal(Socrates)
```

Output

Your program should write only the following to standard output: a numbered list of the input clauses, followed by the proof's resolution steps (one per line), followed by the total number of resolutions performed:

```
1: -Human(x1) | Mortal(x1)
2: Human(Socrates)
3: Animal(F2(x2)) | Loves(F1(x2), x2)
4: -Loves(x2, F2(x2)) | Loves(F1(x2), x2)
5: -Mortal(Socrates)
1 and 2 give 6: Mortal(Socrates)
5 and 6 give 7: <empty>
1646 total resolutions
```

Note the special printing of the empty clause. If you detect that no proof exists, print `No proof exists.` instead of the resolution steps. The other parts of the output should be the same.

Execution

Please use `make.sh` and `run.sh` scripts as usual. We supply:

`fol-proof-validator` takes your program's name (which should be `run.sh`) as its first argument and your program's arguments as its remaining arguments. Passes standard input to your program. Runs your program, parses its output, and verifies that its proof (if any) is valid.

`fol-prove-reference` a sample solution

Write-up

In your final submission, include a brief write-up answering the following questions:

1. Describe any implementation choices you made that you felt were important. Mention anything else that we should know when evaluating your program.
2. What can you say about the the time and space complexity of your program?
3. What suggestions do you have for improving this assignment in the future?

Graduate Extensions

Those in 830 must implement two extensions: preferring smaller clauses during resolution (a generalization of unit preference) and answer literals (for getting constructive proofs). If the user wants a constructive proof, they will include `Ans(x)` (for some variable `x`) in the negated query that you receive as input.

Evaluation

Our emphasis will be on correctness rather than speed.
The graduate student extensions are worth 2 points.

Design Suggestions

Use the example KBs that we distribute to test your code. The examples in the book also make good test cases. You probably want to start by printing out all the resolutions that you do. Once the prover is working, you can implement the necessary record-keeping to just print the proof.