CS200 - Programming Assignment 1: Computer-Assisted Proofs

Read this entire document carefully before starting the assignment!!

Motivation

A current active field of research at the intersection of computer science and mathematics involves using computers to help find proofs. While some work is being done to use AI techniques to write proofs in the way a human does, most computers do "proof by exhaustion," which is when a computer tries all possible cases of interest. For example, for a proof of the Pythagoeran triples problem, a computer had to check a trillion different cases (which took 2 days on a supercomputer). When a proof is created with the help of a computer, we call it a "computer-assisted proof."

In this programming assignment, you will write a program to aid in a computer-assisted proof. It will take in a list of true statements, and one statement whose truth value is unknown (the one you are trying to prove). The program then tells you if you have enough information to prove the unknown statement is true or false, or whether you need more information.

Guidelines

Please read and abide by the honor code guidelines in the syllabus.

Please read the rubric so you know how you will be graded. For example, turning in a program that compiles and runs without errors but does nothing will earn you more points than a program that is close to working but does not compile or contains errors on running.

If you use someone else's code or package, please reference the author and where you found it (both in the beginning comment and at the location in the code). You may use any functions, methods, or packages that you find useful (as long as they do not implement the full assignment). Part of the goal of this assignment is for you to find and learn how to use new functions/methods/packages that are either built-in to the language or external. Thus you should feel free to do an internet search for how to implement some bit of functionality that you desire.

Put a multi-line comment at the beginning of your program. It should contain:

- Your name
- "Programming Assignment 1"
- The name of anyone you worked with and the nature of your collaboration.
- A list of any external code sources.
- Sample output from your program
- The amount of time (approximately) that you spent on this assignment

Assignment

Write a program in Python or Java that takes as input a list of strings. Each string should be a predicate that consists of capital letters ('A', 'B', 'C', ..., 'Z') representing Boolean variable, the operators 'and', 'or', and 'not', and parentheses. The first string in the list is the statement you are trying to prove (the test statement). The rest of the strings in the list are statements that you assume are true (input statements). Your program should output `true`, `false` or `not enough information` depending on whether the test statement is true, not true, or you don't have enough information, given that the input statements are true.

Your program should use an exhaustive search approach. In other words, it should test all possible assignments of True and False to the variables (like we do in a truth table), exclude those assignments that are not consistent with your input statements, and for assignments that are consistent, see if your test statement always has the same truth value.

For example, if the input were `('not A and C', '(A and B) or not C', 'C or B')` then we would be interested in the following truth table:

A	B	C	$(A \wedge B) \vee \neg C$	$C \vee B$	consistent	$\neg A \wedge C$
T	T	T	T	T	yes	F
T	T	F	T	T	yes	F
T	F	T	F	T	no	
T	F	F	T	F	no	
F	T	T	F	T	no	
F	T	F	T	T	yes	F
F	F	T	F	T	no	
F	F	F	T	F	no	

We see that whenever the input statements are true (those rows that have "yes" in the **consistent** column), the value of $\neg A \wedge C$ is false. So in this case, your program should output `false` because the test statement is false whenever the input statements are true. (Some rows of the final column are blank because we don't care about situations when the input statements are not all true.)

You program *should not* output such a truth table, but it should virtually go through the rows of such a table to test the consistency of the input statements and the truth value of the test statement.

Here is a sample output:

```
>>> prove('not A and C', '(A and B) or not C', 'C or B'))
false
>>> prove('not A and not C', '(A and B) or not C', 'C or B'))
not enough information
>>> prove('A or not C', '(A and B) or not C', 'C or B'))
true
```

Tips

For this particular assignment, Python is easier than Java, because of the `eval` method, which takes as input a string containing a Boolean expression and outputs the Boolean value of the evaluated expression. For example, `eval("(True or False) and not False")` returns `True`. Java does not have such a built-in method, but you can use one that someone else has written. (See for example these options.)