Program 2: Deductive Retriever (part 1)
Due date: Tuesday, February 24.

The problem: Once we have unification, we can set up a database of assertions and rules to assert and retrieve first-order sentences. This database will be a set of sentences considered conjunctively, and inference will be done as discussed in class and in Dean et al 3.8.

Assignment: You will develop and test a Deductive Retriever. In the first part, you will build a basic retriever that is “correct” as long as nothing is erased. In the second part, you will add mechanisms that will allow correctness even if things are erased.

(Part 1):

Design a DR that supports the following functions:

- Assert (<formula>)
- Erase (<formula>)
- Retrieve(<formula>)
- ClearDB()
- ClearAsserts()

You should be able to assert or erase:

1. “Facts”: atomic sentences, negations of atomic sentences, or conjunctions of these
2. Backward chaining rules of form (BC <consequent> <antecedent>), where <consequent> is an atomic sentence or the negation of an atomic sentence, and <antecedent> is an atomic sentence, its negation, or a conjunction of these
3. Forward chaining rules of the form (FC <antecedent><consequent>), where <antecedent> is an atomic sentence or the negation of an atomic sentence, and <consequent> is an atomic sentence, its negation, or a conjunction of these.

Whenever you assert a conjunction, it will be equivalent to asserting each conjunct individually.

You should be able to retrieve facts (as in 1.). A retrieve will return a number of substitutions (that is, variable bindings) corresponding to the values of the retrievable things.

Your code should handle negation, and simplify double negation; (not (not (big fred))) should be asserted, retrieved, or erased as (big fred).

ClearDB() should remove all assertions and rules from the database.

ClearAsserts() should remove all non-rule assertions from the database.
Nonmonotonicity and negation

The DR as specified handles negation explicitly, that is, negated formulas are explicitly in, or inferable from, the database. We wish to add a nonmonotonic connective, Consistent, with syntax (Consistent formula) treated like a formula. Its semantics are as follows: (Consistent $\langle$formula$\rangle$) is true (i.e. can be retrieved) if (not $\langle$formula$\rangle$) cannot be retrieved. The mechanism for retrieving such forms will need to be built into Retrieve(). It is not appropriate to assert (Consistent formula); rather this will generally be seen in backward chaining rules.

Assignment specifics: As with the first assignment, your syntax may vary as long as you define it. Also like the first assignment, you will need the capability to read and write to and from files.

Specifically, you should be able to read “commands” (directives to call functions with given arguments) from an input file, then echo the command and the returned result to an output file.

There are a bunch of details that we will cover in class, also lots of examples.

This program can be written in a (small) group if you prefer.

Report due:
1. Specification of the textual syntax of your sentences, including BC, FC, and Consistent.
2. Documented source code, so it could be compiled and run by someone else (hard copy plus electronic version, zipped and submitted to robert via email, named Project2_300_<your name>). Documentation should identify all classes or types defined and used.
3. Test results (include test files in zipped stuff, plus enough hard copy to illustrate).
4. Short discussion of design, including choices made and alternatives, plus a critique of the design (e.g. how could it be improved). In particular, suggest mechanisms that could make it more efficient (and/or identify things you did to improve the efficiency).