6.825 Project 1c

Due: Thursday, October 24, 5PM

Your assignment is to do some proofs using a proof checker. Please read this whole handout carefully before starting.

1 Using the Proof Checker

The Java application `RRPC.Checker` is a resolution-refutation proof checker. It doesn't do proofs on its own, but it will let you enter a proof step-by-step, checking each step as it goes. The proof checker has two main panes (sub-windows): the top one contains clauses that are derived from the given axioms and desired conclusion. The bottom one contains clauses that are derived by doing proof steps.

The checker is always in one of two modes. In *axiom-entry mode*, it allows you to add and delete axioms in various ways. In *proof mode*, it allows you to make proof steps and to delete them. In proof mode, you may not add or delete axioms.

There are additional commands for loading and saving proofs in different formats.

There is set of buttons that are always visible (though some are disabled in different modes), which are explained in the following.

In order to use this tool effectively, you'll have to be familiar with the method for selecting and de-selecting multiple items in Java windows on your platform. It's usually something like shift-click or command-click that allows you to add or delete an item from the current selection set.

1.1 Add Axiom

Prompts the user for a new axiom. Type an axiom into the text field (in the "homework language" of project 1b), and click OK. The axiom will be converted into clausal form and entered as a list of lines in the axiom pane of the proof.

1.2 Load Axioms

Reads in a file (selected via a dialog) and converts it to clausal form; the resulting clauses are entered in the axiom pane. The file can contain multiple sentences in the homework language (they do not all need to be connected with `^` , as in the previous assignment).

1.3 Add Conclusion

This is just like **Add Axiom**, except that it negates the formula entered by the user before converting to clausal form. It's useful for entering the desired conclusion of your proof. It need not be done last.

If you're loading your axioms in from a file (a good idea in general), then you can just include the negation of your desired conclusion, and you don't need to worry about using this button to do anything.

1.4 Delete Clause

This button will delete a clause that has been selected in the axiom pane. If 0 or more than 1 clauses have been selected, it complains.

1.5 Clear Axioms

Deletes all the clauses in the axiom pane.

1.6 Start Proof

Switches into proof mode. All the buttons in this group (for adding and deleting axioms) will be disabled, and the next group of buttons will be enabled. If you want to go back and edit your axioms, you have to use the **Clear Proof** button, which will move you back to axiom-entry mode.

1.7 Resolve

If a total of two lines are selected from the axiom and/or proof panes, then this button will run resolution on those two lines. If there is more than one possible result, the user will be offered all possible results to choose from in a dialog box. The resulting clause is entered into the proof pane.

1.8 Factor

If a single line is selected from the axiom or proof pane, this button will run factoring on that line. Factoring is an inference rule that looks to see if can unify two literals within the same clause. If it can, it deletes one, and applies the resulting substitution to the rest of the clause. It's useful for situations when you have duplicate literals, or, for example, to derive $\mathsf{P}(\mathsf{c},\mathsf{c})$ from $\mathsf{P}(\mathsf{X},\mathsf{c}) \vee \mathsf{P}(\mathsf{c},\mathsf{Y})$.

1.9 Paramodulate

If a total of two lines are selected from the axiom and/or proof panes, then this button will run resolution on those two lines. Doing a paramodulation step will typically require a number of interactions with the user:

- First, if there are multiple positive `Equals` literals in the selected clauses, the user is asked to select one to use as the equality in the paramodulation.
- Next, the user is asked which of the terms in the equality should be matched into the other sentence.
- Finally, if there are multiple terms in the other sentence that match the selected term, the user is ask which one to use (to serve as `r` in the paramodulation rule).

The resulting clause is entered into the proof pane.

1.10 Delete Step

Watch out for this one! If there is a single line selected in the proof pane, this button will delete that line *and all further lines that depend, directly or indirectly, on this line.*

1.11 Clear Proof

Deletes all the proof steps and goes back into axiom-entry mode.

1.12 Save Proof

Saves the current proof out to a file (selected by the user via a file-selection dialog) using the Java serialization methods. *This file is not human-readable and should not be turned in to TAs.* Files written this way may be read in by **Load Proof**, but not by **Load Axioms** (which reads a text file).

I'm currently having trouble making this work between sessions on the same machine. Don't rely completely on it. (Use Output Proof to make a human-readable backup of your partial proofs).

1.13 Load Proof

Deletes all of the current axioms and proof steps. Replaces them with the proof contained in the file (selected via a dialog). *This method can only load files that were generated by* **Save Proof**.

1.14 Output Proof

Writes a text file (selected via dialog) containing all of the current axioms and proof steps in a human-readable form, suitable for handing in. This file format is not currently readable as input to the proof checker.

1.15 Quit

Quits.

2 Tasks

For each task, hand in a proof (generated using the **Output Proof** button of the proof checker).

1. (Warmup) Write axioms to formalize the following argument:

   - All men are mortal.
   - All mortals are boring.
   - Hera is not boring.
   - Therefore, there exists someone who is not a man.

   Then prove that the conclusion follows.

2. The file `siblings.txt` contains axioms that attempt to formalize the riddle

Sisters and brothers have I none; yet that man's father is my father's son.

Who is that man? The axioms make a particular assertion about who that man is (it's negated, in the last line of the file). Use RRPC to prove that the assertion is true.

3. The file `blocks.txt` contains axioms describing a blocks-world planning problem in situation calculus. The last line says

   ```
   ~exists s on(b,c,s) ^ ~answer(s)
   ```

   This is an old trick for finding out which situation it is that satisfies our requirements. Thus, in doing your proof, you should stop when you have a line with the single literal `answer(α)`. The term `α` will name a situation as a function of how it results from doing a sequence of actions, starting in `S0`.

   Prove that a situation in which `b` is on `c` exists.

   Extra credit: If you're feeling enthusiastic, replace the last line with

   ```
   ~exists s on(a,b,s) ^ on(b,c,s) ^ ~answer(s)
   ```

   and prove that there's a situation satisfying this requirement, as well.

4. The file `trains1.txt` contains axioms with layout 2 and the original wrong definition of *legal* from project 1b. Some of the axioms are slightly changed.[1] The last line of the file is

   ```
   ~(on(T1,R1,S1) ^ on(T2,R3,S1) -> legal(T1,R1,R2))
   ```

   The idea is that we want you to prove that if `T1` is on `R1` in `S1`, and `T2` is on `R3` in `S1`, it is legal for `T1` to move to `R2`.

   The next problem is kind of hard, and so this is a warm-up for that one (it has the same axioms, but a different conclusion). Be sure you can do this one first (you'll learn a strategy that's necessary for the next one).

5. The file `trains2.txt` contains the same axioms as the previous problem, but the conclusion is

   ```
   ~(on(T1,R1,S1) ^ on(T2,R3,S1) -> ~safe(S2))
   ```

   Now, we're asking you to show that, for this particular choice of initial conditions, the second situation is not safe.

[1] We had to change the axioms around to handle non-determinism a bit differently. In the new version, we say that each train is on exactly one rail in `S1`. But we allow trains to be on more than one rail in `S2`, using that ability to express the non-determinism in the trains' motion. You can read the *on* predicate as being something more like "could possibly be on."

3 Advice

The last three of these problems are pretty big. They're much too big to just start doing proof steps and hope you get to the right place. You have to have a plan. The best way, I've found, to have a plan, is to do an informal proof on paper, using natural deduction steps, rather than resolution. Think of how you would use the axioms to argue that the conclusion follows, if you were asked to do so for a math or theory class. Then, given that proof, try to do the same thing using resolution. Spend some time understanding which clauses go with which original axioms, and what they mean.