Automated Theorem Prover

Ranveer Aggarwal 120050020
Dheerendra Rathor 120050033
Adit Kabra 120050034

Guide: Dr. Pushpak Battacharyya
IIT Bombay

March 01, 2015

Aim

Here, we build an automated theorem prover to search for proofs of arbitrary formulae under the axioms of **Propositional Calculus** as an assignment of the course CS 386.

Propositional Calculus

Propositions stand for fact/assertions; they're declarative statements as opposed to imperative or interrogative statements. The only operators we have are $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT) and $\implies$ (IMPLICATION), wherein $\neg$ and $\implies$ form a minimal set.

Hilbert's Formalisation of Propositional Calculus

- Elements are propositions : Capital letters
- Operator is only one : $\rightarrow$ (called implies)
- Special symbol F (called 'false')
- Two other symbols : '(' and ')'
- Well formed formula is constructed according to the grammar
  $WFF \rightarrow P|F|WFF \rightarrow WFF$
- Inference rule : only one
  Given $A \rightarrow B$ and A, we can say B
  This is called Modus Ponens
- Lastly, there are some starting structures in the form of axioms

Hilbert's Axioms for Propositional Calculus

The following Axioms for Propositional Calculus are given:

A1 $(A \to (B \to A))$

A2 $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

A3 $(((A \to F) \to F) \to A)$

We'll be using the axioms as A1, A2 and A3 throughout this presentation (they're to be referred from this slide).

Deduction Theorem

By using Deduction Theorem (DT), we prove that a proof for the system exists, without actually proving the statement. Basically, DT is a meta-theorem. This is how it goes:
If $A_1, A_2, ...A_n \vdash B$
then $A_1, A_2, ...A_{n-1} \vdash A_n \rightarrow B$
where $\vdash$ is read as "derives".

Mechanics of a Propositional Calculus Proof

1. Sequence of well formed formulae
2. Start with a set of hypotheses
3. The expression to be proved should be the last line in the sequence
4. Each intermediate expression is either one of the hypotheses or one of the axioms or the result of modus ponens
5. An expression which is proved only from the axioms and inference rules is called a THEOREM within the system

Working of our Theorem Prover

Our theorem prover basically uses the deduction theorem to show that the proof for a particular statement does exist. Let us say the input statement is $A_1 \rightarrow A_2 \rightarrow ... \rightarrow A_n$.

- We write it as $\vdash A_1 \rightarrow A_2 \rightarrow ... \rightarrow A_n$
- Then we bring each term to the LHS, one by one. Hence, we get something like $A_1, A_2, A_3, ...A_{n-1} \vdash A_n$
- Lastly, we're left with $A_1, A_2, A_3, ...A_{n-1}, \neg A_n \vdash F$ (We say it's sufficient to prove this to prove point 1; the proof of this is trivial, try taking the terms one by one beginning from the last to th RHS to get back the statement in point 1)
- Now, since $F \vdash F$, we try to get F by using the Propositional Calculus Mechanics on $A_1, A_2, ...A_n$, getting which, the proof stands complete.

Code Working

- The theorem prover has been programmed in Python
- We first get a statement as input in the form of $A \wedge B \vee \rightarrow \neg C \ldots$
- We then reduce it to the form of $A_1 \rightarrow A_2 \rightarrow \ldots \rightarrow A_n$, where each of $A_1, A_2, ..$ are composed of statements themselves
- We then get the hypotheses using deduction theorem and search for F in it. If found, we return *True*, meaning that the theorem is provable

Code Working

- If F isn't found, then we try to apply modus ponens between every pair of statements. Again, we look for an F. If found, we return *True*
- If F still isn't found, we ask the user for axioms in the form of A1, A2 or A3 and also, what variable each of A, B and C in the axioms map to (check the slide on Hilbert's axioms).
- If F still isn't found, we ask the user for yet another axiom and continue until the theorem is proved. The user also has an option to input a theorem outside of the basic three (say the user has proved such an axiom himself/herself on paper).
- If the user gives up, we say that the theorem couldn't be proved.

Conclusion

Hence, our theorem prover does all the mechanical steps which, would be frustrating and time taking for a human being (assuming the number of hypotheses is high). But what the prover cannot do, are the intelligent steps, where it needs human intervention. We could always implement search algorithms that would help the prover to find th required hypothesis to use itself, but programming a perfect search algorithm is quite difficult and surely, the algorithm *will* get stuck at times even if this is done.

Credits

- Lectures by Dr. Pushpak Bhattacharyya
- The Python Documentation

Questions and Improvements

In case of any suggestions/improvements/question, shoot a mail to
ranveer [at] cse [dot] iitb [dot] ac [dot] in or
dheerendra [at] cse [dot] iitb [dot] ac [dot] in or
kabraadit [at] cse [dot] iitb [dot] ac [dot] in.
Thank you!