Applied Automated Theorem Proving

CSE 291-F

Instructor: Sorin Lerner

Some history

- The field of automated theorem proving started in the 1960s
  - SAT and reduction to SAT (early 60s)
  - Resolution (Robinson 1965)
  - Lots of enthusiasm, and many early efforts
  - Was considered originally part of AI
- In the 70s
  - some of the original excitement settles down, with the realization that "interesting" theorems are hard to prove automatically.

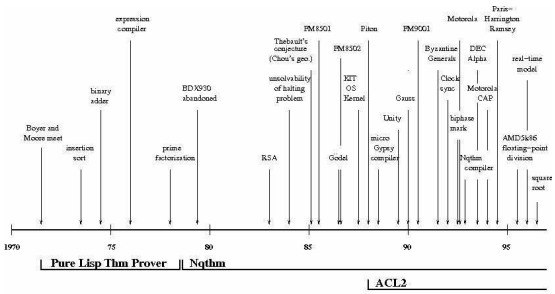
Over the next three decades

- Many large theorem proving systems are born
  - Boyer-Moore (1971)
  - NuPrl (1985)
  - Isabelle (1989)
  - Coq (1989)
  - PVS (1992)
  - Simplify (1990s)
- The list of theorems proven automatically/semi-automatically grows

On the math side

- 1976: Appel and Haken prove the four color theorem using a program that performs a gigantic case analysis (billions of cases).
- First use of a program (essentially a simple "theorem prover") to solve an open problem in math. The proof was controversial and attracted a lot of criticism.
- Other open problems have since been solved using theorem provers.

On the verification side



And this is just one theorem prover!

And yet…

- In 1979, DeMillo, Lipton and Perlis, in a now famous paper, argue that software verification is doomed.
- Why?
  - Too hard to verify by hand
  - Too hard to verify automatically
  - Nobody will check your verifications anyway
  - As opposed to math, where a proof becomes a proofs only after it has been validated by the community

And then… the internet happens

- Amount of code exposed to malicious attacks skyrockets
  - Vulnerabilities are widely exploited
  - And are worthy of the NYTimes front page
- At the same time, state of the art improves
- Result: technological readiness + increased cost of bugs leads to a renewed interest in software verification, and the use of analysis techniques and/or theorem provers to verify software

Recent uses of theorem provers

- ESC/Java: pre- and post-condition checker for Java
  - ESC/Java is a tool to check that the pre-condition of a method implies the post-condition of the method. Underneath the covers, ESC/Java uses a fully automated theorem prover.
- SLAM: verifying C software
  - SLAM verifies that C programs adhere to API usage rules, such as “a lock can be released only if it was previously acquired”, or “a file can be written to only if it was previously opened”. SLAM uses a theorem prover to perform predicate abstraction.

Recent uses of theorem provers

- Verisoft: end-to-end correctness
  - This porject uses interactive theorem provers to show the correctness of the software itself, but also of all the artifacts needed to execute the software (e.g. hardware and compiler).
- Rhodium: automatically proving compilers correct
  - Rhodium is a language for writing compiler optimizations that can be proven correct automatically using a theorem prover.

This course

- Learn about ATPs and ATP techniques, with an eye toward understanding how to use them in parctice
  - Look at recent successful uses of theorem provers, and try to learn from them
  - Understand how ATP techniques work, and what the tradeoffs are between techniques
  - Understand how ATP techniques can be applied in the broader context of reasoning about systems
- Apply what you've learned in a course project

Course topics

- Search techniques
  - semantic domain, proof domain
- Handling
  - equality, quantifiers, induction, decision procedures
- Applications
  - ESC/Java, SLAM, Rhodium, proof carrying code
- Understanding how all of the above interrelate

Course work

- Low credit option: 1 credit
  - Attend class, participate in discussions; graded S/U
- Medium credit option: 2 credits
  - Also read papers, write paper reviews; graded S/U
- High credit option: 4 credits
  - Also work on the course project

Course work

- I would really encourage you to take the course for 4 credits
  - the project is the funnest part!
- If you are busy, there are many ways to make the project still work
  - combine with your current research project
  - combine with a project from another class
- I would like to meet with each of you one-on-one for 5-10 mins, just to get a feeling for what you're interested in

Course project, part I: mini-project

- Given a problem, you are asked to encode it in two (maybe three?) theorem provers
- Written report stating what worked, what didn't, and what the differences were between the various theorem provers
- Probably due around Feb 3rd

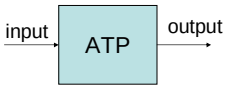
Course project, part two: the real thing

- Groups of 2
- Apply theorem proving technology to a problem of your choice
  - start thinking about topics now
- Milestones:
  - project proposal
  - mid-term presentation to the class
  - 2 meetings throughout the quarter with me
  - final presentation to the class

Course pre-requisites

- Undergrad level logic
  - We won't be doing any hardcore math…
  - But we will talk about predicate logic, first order logic, and proofs by induction
- Some familiarity with functional programming
  - mini-project will require you to write some LISP code
- Both of the above are usually covered in a standard undergrad cs curriculum
  - Talk to me if you think you don't have the pre-requisites

What is an automated theorem prover?



Input: Example theorems

- Pythagoras theorem: Given a right triangle with sides A B and C, where C is the hypotenuse, then $C^2 = A^2 + B^2$
- Fundamental theorem of arithmetic: Any whole number bigger than 1 can be represented in exactly one way as a product of primes

Input: Example theorems

- Pythagoras theorem: Given a right triangle with sides A B and C, where C is the hypotenuse, then $C^2 = A^2 + B^2$
- Fundamental theorem of arithmetic: Any whole number bigger than 1 can be represented in exactly one way as a product of primes

Input 1: Theorem

- Theorem must be stated in formal logic
  - self-contained
  - no hidden assumptions
- Many different kinds of logics (first order logic, higher order logic, linear logic, temporal logic)
- Different from theorems as stated in math
  - theorems in math are informal
  - mathematicians find the formal details too cumbersome

Input 2: Human assistance

- Some ATPs require human assistance
  - e.g.: programmer gives hints a priori, or interacts with ATP using a prompt
- The harder the theorem, the more human assistance is required
- Hardest theorems to prove are “mathematically interesting” theorems (eg: Fermat’s last theorem)
- In this course, will be dealing with theorems about software artifacts – mathematically uninteresting, but practically useful.

Output

- Can be as simple as a yes/no answer
- May include proofs and/or counter-examples
- These are formal proofs, not what mathematicians refer to as proofs
- Proofs in math are
  - informal
  - “validated” by peer review
  - meant to convey a message, an intuition of how the proof works -- for this purpose the formal details are too cumbersome

Output: meaning of the answer

- If the theorem prover says “yes” to a formula, what does that tell us?
  - Soudness: theorem prover says yes implies formula is correct
  - Subject to bugs in the Trusted Computing Base (TCB)
  - Broad defn of TCB: part the system that must be correct in order to ensure the intended guarantee
  - TCB may include the whole theorem prover
  - Or it may include only a proof checker
  - Does it include the human-provided hints?

Output: meaning of the answer

- If the theorem prover says “no” to a formula, what does that tell us?
  - Completeness: formula is correct implies theorem prover says yes
  - Or, equivalently, theorem prover says no implies formula incorrect
  - Again, as before, subject to bugs in the TCB

Output: meaning of the answer

- A sound and complete ATP is essentially a decision procedure for the input logic.
- For expressive logics, cannot be sound and complete
- Consider the theorem: “My program is free of buffer overruns”. If you had to choose between soundness and completeness, what would you choose? Why?

Output: meaning of the answer

- ATPs first strive for soundness, and then for completeness if possible
- Many ATPs are incomplete: “no” answer doesn’t provide any information
- Many subtle variants
  - refutation complete
  - complete semi-algorithm

For Thursday

- Read DeMillo, Lipton and Perlis
- Read Moore’s talk from Zurich 05
- Very high level, easy read. No need to write reviews.
- We will discuss these papers, and then we will start talking about logics, and how to encode problems in logics