Course Information

Prerequisites:
Logic in Computer Science (taught last semester) or equivalent knowledge of first-order logic

Contents of the Course
In this course we look at automated deduction within the context of first-order logic. We cover automated theorem proving techniques like resolution and rewriting. We also look at first-order theories for which the decision problem is decidable. We will cover a variety of problem domains and techniques and will also look at some applications.

Webpage:
http://www.cs.nyu.edu/courses/spring09/G22.3033-010/index.html

Email List
http://www.cs.nyu.edu/mailman/listinfo/g22_3033_010_sp09

Outline

• Course Information
• History and Motivation
• Objective Caml (OCaml)
• Symbolic computation and OCaml

Sources:

Course Information

Sources:
The primary source for the class is an unpublished book by John Harrison called “Introduction to Logic and Automated Theorem Proving.” This and other source materials will be made available to those in the class.

Exams
There will be no exams.

Assignments:
There will be periodic assignments throughout the semester.

Project:
In addition to assignments there will be a final project which will require a significant amount of programming effort.

Grading:
Assignments: 50%, Project: 50%
Course Information

Academic Integrity

Please review the departmental academic integrity policy.

In this course, you are expected to do your own work. Discussing assignments with classmates is acceptable, but all code submitted must be your own. Any help you receive must be clearly noted on your assignment. Also, you should consult the instructor before using materials or code other than that provided in class. Copying code or other work without giving appropriate acknowledgment is a serious offense with consequences ranging from no credit to potential expulsion.

Acknowledgments

In addition to Harrison’s book, I will draw from a number of other sources throughout the semester. Each lecture will include a list of source material.

Why Automated Reasoning?

From very early on, philosophers have dreamed of machines that can reason.

Leibniz (1646-1716), who with Newton generally shares credit for inventing Calculus, first proposed the ambitious goal of mechanizing the process of human reasoning, saying,

Once this is done, then when a controversy arises, disputation will no more be needed between two philosophers than between two computers. It will suffice that, pen in hand, they sit down to their abacus and (calling in a friend if they so wish) say to each other: let us calculate.

In 1945, Dr. Vannevar Bush, director of the Office of Scientific Research and Development, wrote the following:

Logic can become enormously difficult, and it would undoubtedly be well to produce more assurance in its use. ... We may some day click off arguments on a machine with the same assurance that we now enter sales on a cash register.

From As We May Think, http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm

Mathematicians have long understood the importance of proofs as an assurance of the correctness of a logical argument.

Many modern proofs in mathematics are very difficult, and could benefit immensely from automated proof assistance.

In 1998, Tom Hales gave a proof of a famous unsolved problem known as the Kepler conjecture. The proof was so long and complicated that it defied all human efforts to check it.

As a result, Hales began the Flyspeck project, an effort to automate the proof of the Kepler conjecture (see http://code.google.com/p/flyspeck/).

More recently, the need for proofs in computer science applications has become urgent. Proofs provide assurances of correctness that cannot be obtained using simulation and testing.

Proof-based verification techniques are now standard in hardware design and are becoming more and more prevalent in software design as well.
History

Logic itself dates back to the 4th century BC with Aristotle's attempt to analyze sound reasoning.

Leibniz had great ambitions for symbolic logic, but did not get very far himself.

The subject really blossomed in the 19th century with the work of Boole, De Morgan, Dedekind, Frege, Peano, Pierce, Cantor, and others.

The 20th century brought great minds like Hilbert, Skolem, Herbrand, Gödel, Gentzen, and Church who laid the foundation for modern mathematical logic.

The very first computer-generated mathematical proof was produced here at New York University using a program written by Martin Davis in 1954. The program was an implementation of a decision procedure for Presburger Arithmetic.

Its great triumph was to prove that the sum of two even numbers is even.

In 1957, Newell, Shaw, and Simon, researchers at RAND created a program called the Logic Theory Machine (LT).

The research...is aimed at understanding the complex processes (heuristics) that are effective in problem-solving...we are not interested in methods that guarantee solutions...Rather, we wish to understand how a mathematician, for example, is able to prove a theorem even though he does not know when he starts how, or if, he is going to succeed.

LT introduced many important ideas such as proof-trees, subgoals, and an early version of unification, and was successfully able to verify theorems from Russell and Whitehead's Principia Mathematica.

In 1960, Hao Wang published a paper reporting that many of the LT proofs as well as others from Russell and Whitehead were easily decidable. He was able to prove hundreds of these theorems in minutes.

The most interesting lesson from these results is perhaps that even in a fairly rich domain, the theorems actually proved are mostly ones which call on a very small portion of the available resources of the domain.

Wang vs Newell-Shaw-Simon

On the contrasting approaches of Wang and Newell et al., Martin Davis wrote,

The controversy referred to may be succinctly characterized as being between the two slogans: “Simulate people” and “Use mathematical logic”. Although this controversy has generated much heat, there has never been much doubt among serious workers in the field that both streams of ideas were important...Thus as early as 1961 Minsky remarked:

it seems clear that a program to solve real mathematical problems will have to combine the mathematical sophistication of Wang with the heuristic sophistication of Newell, Shaw and Simon.

In 1965, Robinson proved that the simple resolution rule was a sound and complete inference procedure for first-order logic. Much of the work in automated deduction since then has been based on this rule.

However, for a given first-order theory, resolution may do poorly compared with domain-specific decision procedures.

Wang observed:

That proof procedures for elementary logic can be mechanized is familiar. In practice, however, were we slavishly to follow these procedures without further refinements, we should encounter a prohibitively expansive element. ...In this way we are led to a closer study of reduction procedures and of decision procedures for special domains.

In this course, we will look at some of these decision procedures for special domains, with an eye towards efficiency and practicality.
John Harrison’s Book

John Harrison is a well-known researcher in automated deduction, a researcher at Intel, and the author of the HOL Light theorem prover.

For the past 15 years or so, as a side project, he has been working on a book about automated deduction.

He has graciously given consent for us to use an unpublished version of the book.

Some pros and cons of the book:

- **Pro**: covers a lot of topics with many good references
- **Pro**: an accompanying set of Objective Caml source files containing implementations of most of the algorithms discussed
- **Con**: sometimes the text mixes theory and implementation in a way that can be hard to follow

For our purposes, the advantages far outweigh the disadvantages. To mitigate the problem of the book’s occasionally confusing explanations, I will try to always separate theory and implementation in the class discussions and notes.

Objective Caml

Objective Caml (OCaml) is a general purpose functional programming language developed at INRIA. All of your programming assignments will be done in OCaml.

OCaml is available from http://www.ocaml.org

Some of the useful resources available from this page include

- OCaml manual (the *Index of Values* is especially helpful).
- Online tutorials

There are various OCaml modes for emacs. I happen to like the *tuareg* OCaml mode available at http://www-rocq.inria.fr/~acohen/tuareg/.

If you have access to a machine on which you can install OCaml, I suggest you do so. It is also available on the CIMS linux machines.

OCaml

We will spend some time with Jason Hickey’s introduction to Objective Caml, available on the web at:


Harrison’s OCaml Libraries

John Harrison has created a superb platform for playing with automated theorem proving in OCaml. A complete set of files is available from his webpage at http://www.cl.cam.ac.uk/~jrh13/atp/.

However, I will be providing a modified version of these files that are easier to use interactively. They will be available on the webpage.

We will start by looking at the following files:

- startup.ml (loads other files)
- lib.ml (general purpose routines)
- intro.ml (simple examples of embedding logic in OCaml)
Some Useful OCaml Routines

Directives

• `#use "filename.ml"`
  Load filename.ml as if it had been typed into the toplevel.

Built-in operations

• `+`, `+-`, `+/
  Addition of integers, floats, and Nums

• `@`
  List concatenation

• `^`
  String concatenation

• `failwith "message"
  Raise a Failure exception with the given message.

From lib.ml

• `**`
  Function composition.

• `hd l`, `tl l`
  First element of a list l, everything but the first element.

• `itlist f l b`
  Iterate over list l (from last element to first), applying binary function f. b
  provides the initial second argument to f.

• `end_itlist f l`
  Same as above, except start with the last element of l instead of b.

• `partition p l`
  Divide list l into a pair of lists based on predicate p.

• `filter p l`
  Keep only elements of list l which satisfy predicate p.

From lib.ml (continued)

• `length l`
  Length of list l

• `find p l`
  Return the first element of l satisfying predicate p.

• `map f l`
  Return a new list formed by applying function f to every element of l.

• `smap f l`
  Like map, but duplicates are removed

• `allpairs f a b`
  Return a new list formed by applying binary function f to every pair of
  elements from lists a and b.

• `assoc a l`
  l must be a list of pairs. assoc a l returns the second part of the pair whose
  first part is a.

In general, to find the definition of a routine, you should look in Harrison’s files
first, and then check the “Index of Values” in the OCaml manual.