An OCaml-based automated theorem-proving textbook

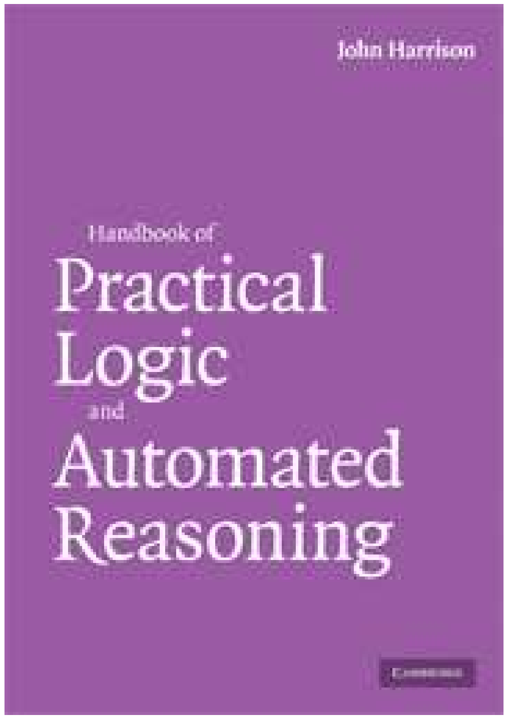
John Harrison, Intel Corporation

Portland Functional Programming Study Group

Mon 11th May 2009 (19:00)

Book plug

Surveys many parts of automated reasoning, via actual code.



Available online from the usual places. Hopefully will be in Powells in a couple of weeks...

Why I wrote this book

Standard existing books on automated theorem proving mostly fall into one of these categories:

- Chang and Lee, *Symbolic Logic and Mechanical Theorem Proving*. Excellent textbook, but mainly focused on resolution with nothing on arithmetic decision procedures, and somewhat dated.

Why I wrote this book

Standard existing books on automated theorem proving mostly fall into one of these categories:

- Chang and Lee, *Symbolic Logic and Mechanical Theorem Proving*. Excellent textbook, but mainly focused on resolution with nothing on arithmetic decision procedures, and somewhat dated.
- Gordon and Melham, *Introduction to HOL: a theorem proving environment for higher order logic*. Fine description and manual for *usage* of HOL system, but entirely focused on that one system and with little explanation of *how* it works.

Why I wrote this book

Standard existing books on automated theorem proving mostly fall into one of these categories:

- Chang and Lee, *Symbolic Logic and Mechanical Theorem Proving*. Excellent textbook, but mainly focused on resolution with nothing on arithmetic decision procedures, and somewhat dated.

- Gordon and Melham, *Introduction to HOL: a theorem proving environment for higher order logic*. Fine description and manual for *usage* of HOL system, but entirely focused on that one system and with little explanation of *how* it works.

There are essentially no textbooks on automated theorem proving covering a wide range of the subject without many prerequisites.

Approach of my book

- In principle, assumes almost no prerequisites, but discusses logic from the very beginning. Does require basic mathematics and functional programming (but these are summarized in appendices).

Approach of my book

- In principle, assumes almost no prerequisites, but discusses logic from the very beginning. Does require basic mathematics and functional programming (but these are summarized in appendices).

- Logical theory and automated theorem proving are explained in a closely intertwined manner. Results in logic are developed, wherever possible, in an explicitly computational way.

Approach of my book

- In principle, assumes almost no prerequisites, but discusses logic from the very beginning. Does require basic mathematics and functional programming (but these are summarized in appendices).
- Logical theory and automated theorem proving are explained in a closely intertwined manner. Results in logic are developed, wherever possible, in an explicitly computational way.
- Methods are explained with reference to actual concrete implementations in OCaml, which readers can experiment with if they have convenient access to a computer.

Nice Knuth quote . . .

“For three years I taught a sophomore course in abstract algebra for mathematics majors at Caltech, and the most difficult topic was always the study of “Jordan canonical forms” for matrices. The third year I tried a new approach, by looking at the subject algorithmically, and suddenly it became quite clear. The same thing happened with the discussion of finite groups defined by generators and relations, and in another course with the reduction theory of binary quadratic forms. By presenting the subject in terms of algorithms, the purpose and meaning of the mathematical theorems became transparent.”

. . . Knuth quote continued

"Later, while writing a book on computer arithmetic [...], I found that virtually every theorem in elementary number theory arises in a natural, motivated way in connection with the problem of making computers do high-speed numerical calculations. Therefore I believe that the traditional courses in number theory might well be changed to adopt this point of view, adding a practical motivation to the already beautiful theory."

More self-justification

Other recent-ish mathematics books take a similar approach:

- Cox, Little and O’Shea, *Using algebraic geometry*
- Kreuzer, *Computational commutative algebra*
- Rydeheard and Burstall, *Computational category theory*

More self-justification

Other recent-ish mathematics books take a similar approach:

- Cox, Little and O'Shea, *Using algebraic geometry*
- Kreuzer, *Computational Commutative Algebra*
- Rydeheard and Burstall, *Computational category theory*

If even category theory can be presented computationally, surely mathematical logic deserves to be:

- Analysis of computation by Turing and others was specifically designed to address decidability questions in logic
- Problems in formal verification are motivating the development of automated theorem provers.
- Logic plays increasing role in programming language design.

Table of contents

Why in a functional language?

Compare another implementation-oriented book, Monty Newborn's *Automated Theorem Proving: Theory and Practice*. What does a functional language give us?

Why in a functional language?

Compare another implementation-oriented book, Monty Newborn's *Automated Theorem Proving: Theory and Practice*. What does a functional language give us?

- High level of data structures, abstraction from machine features like memory allocation: as good as pseudocode yet executable.
- Interactive toplevel very convenient for experimentation.
- Purely functional programming allows exploration without worrying about accidental side-effects.

Why in a functional language?

Compare another implementation-oriented book, Monty Newborn's *Automated Theorem Proving: Theory and Practice*. What does a functional language give us?

- High level of data structures, abstraction from machine features like memory allocation: as good as pseudocode yet executable.
- Interactive toplevel very convenient for experimentation.
- Purely functional programming allows exploration without worrying about accidental side-effects.

On the negative side:

- Less familiar to many readers than C, Java, pseudocode . . .

Why OCaml?

I started using CAML Light when I rewrote the HOL theorem prover in it to give ‘HOL Light’. Since then I’ve stayed on the “upgrade path” via CAML Special Light to OCaml and been quite satisfied.

- I don’t use any exotic features like object orientation, labelled arguments etc.
- Except for printing status messages, all code is completely functional, in traditional ML style.
- I do use automatic prettyprinting and (via camlp4/camlp5) quotation parsing to make interactive exploration convenient.

Only significant complaint about OCaml was the ineptly managed transition to a new, incompatible and completely undocumented version of camlp4.

Versions in other languages?

I’d love to have versions of the code in multiple functional languages. Since I use a ‘lowest common denominator’ subset, this shouldn’t be too bad.

An F# version is almost trivial, because the core language is so close to OCaml. The only exception is parsing and printing, which would need to change.

Sean McLaughlin and Roland Zumkeller have worked on a Haskell port, and have already done a significant portion. Toplevel seems less efficient, but compiled version works very well.

It would be interesting to try rewriting some of the code in a more ‘thematic’ Haskell style using laziness, list comprehensions etc.