Book reviews

Handbook of Practical Logic and Automated Reasoning, by John Harrison, Cambridge University Press, 2009 ISBN 9780521899574
doi:10.1017/S0956796811000220

The steady rise in interest in formal verification of software has naturally been accompanied by increased interest in the theory and tools of verification. While there are many textbooks of logic as well as user guides for the vast array of reasoning tools (automated and otherwise), much less is written about how to bridge the two, never mind good guides to the design problems of building reasoning tools. Harrison's *Handbook of Practical Logic and Automated Reasoning* explores that gap with flair.

Even a casual glance at this hefty 700 page book will quickly dispel any "Handbook" impression which the title disingenuously gives: not a handbook at all, but rather a proper *textbook*. The style, far from that of a *reference* book, is deeply pedagogical, complete with a significant set of well-crafted exercises. It is well written using lucid prose, with the author taking great pains to elucidate important points, pausing to illustrate some subtleties, as well as being rather erudite in its coverage of the relevant literature, both historical and contemporary. As a textbook style introduction to the theory and practice of (first order, mathematical) logic and automated reasoning, it succeeds beautifully.

This textbook masterfully weaves together theory and practice, by interleaving the development of the theoretical underpinnings of each topic covered with fairly well crafted code which implements the constructive portions of each chosen topic. In other words, this textbook is also a *literate program*, with all code written in OCaml, available from Harrison's web site. This really forces the author to cover many topics which are often, regrettably, omitted from other "practical" introductions to logic. Readers primarily interested in the theory covered by this book may find this aspect distracting, but I rather view this aspect of the book as one of its more endearing characteristics. The prose aspects of this *literate textbook* could easily serve as a model for years to come. The code is written using a fairly small subset of the Objective Caml language (no objects, Functors, higher rank polymorphism, nor polymorphic variants). While higher-order functions abound, the author nevertheless does not make heavy use of a more combinator-driven style which is more prevalent in the current functional literature. Eschewing Functors also means that the higher abstractions frequently seen in modern functional programs are also not used. While such a choice does broaden the audience which can easily understand the code, it would nevertheless be quite interesting to redevelop the same code, first in idiomatic Haskell, and then in Agda. But doing this would surely result in a research-level monograph, which was clearly not the author's intent. In other words, while this text does not use the most modern functional programming idioms, this was clearly an explicit decision, which makes sense in the context of the audience for this book.

The set of topics covered might seem somewhat eclectic: propositional logic, first-order logic, equality reasoning, decidable problems, interactive theorem-proving and the limitations of all these approaches. In particular, "decidable problems" and "interactive theorem-proving" have traditionally been seen as belonging in different universes by the more "pure" adherents of formal reasoning. Of course, there have always been systems which eschewed this particular balkanisition, with ACL2, IMPS and PVS immediately coming to mind, and Harrison's own HOL Light following in the tradition. But this is now changing, as traditionally very pure

systems are embracing the idea that efficient computation is not necessarily anathema to correctness. The decision procedures shown here fall in an interesting gray area: they are fully *justified* (usually by semantic, model-theoretic arguments, unlike the more syntactic style frequently adopted elsewhere) as well as type-safe, but not fully *proven* to be correct. In most cases, the results of the algorithms are easy to formally verify, and thus provable correctness is not strictly necessary.

Each chapter of this book very carefully covers the necessary theory and motivates a systematic development of the algorithms which make the theory *practical*. Reading the whole book amply justifies this choice of topics, and presents a solid holistic picture of what is covered. One can't help but wish for the author to write a follow-up treatise on higher-order logic in the same vein, seeing the author's *HOL Light* is a rather successful proof assistant based on higher-order logic. To this day, Harrison still holds the record for the largest percentage of the *Top 100 mathematical theorems*[1].

It would be easy to imagine using this book for several courses on logic, theorem proving, and decision procedures, from an introductory course to several Master's level advanced courses. There is in fact so much material here that one could probably create a whole stream (say of 4 courses) in an undergraduate curriculum for specializing in mechanized mathematics. Any researcher who wants to learn how simple theorem proving systems are built would greatly benefit from reading this book. This deserved praise comes with a caveat: implementing "real" systems is a significantly more subtle business than the elegant development of the varied algorithms presented here. This is not so much a criticism of the book itself, but a warning to readers who might feel they have learned so much from this book (and they will undoubtedly learn a lot from it), that they are essentially an expert in the topic, which would certainly not be the case.

Seen as a textbook for a variety of courses which could be taught in mathematics, logic and computing, at the advanced undergraduate as well as at the beginning graduate level, this monograph excels. This reviewer unreservedly recommends it for this purpose. Seen as a wonderful tutorial for a researcher in functional programming to learn the practice (and theory) of applicable mathematical logic and the design of reasoning procedures, this is surely the most approachable such text. But the reader should be well aware that while some subtle issues are well-covered, others (like deeper issues regarding syntax and semantics, intensional and extensional reasoning, trustability of decision procedures written outside of a logic, the fine line between deduction and computation) which are crucial for a deep understanding of the modern issues facing mechanized mathematics, cannot be covered in a textbook at this level. So why even bother to mention that in a review? Mostly because this book is such a pleasure to read, and covers such a wealth of fascinating and diverse material, that it is very easy to lose sight of the fact that this is not a comprehensive research monograph. So, please, go read it and enjoy!

JACQUES CARETTE
carette@mcmaster.ca

How to think about algorithms, by Jeff Edmonds, Cambridge University Press, ISBN 0521614104
doi:10.1017/S0956796811000177

How to Think About Algorithms (HTTAA) is a textbook written by Professor Jeff Edmonds to teach students what is the best approach to think about algorithms abstractly. The

[1] `http://www.cs.ru.nl/~freek/100/`