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/`

intended objective is that after reading the book the reader should be able to use the tools presented in the book to tackle new algorithmic problems with confidence. When writing the book, the author explicitly avoided the creation of *yet another dictionary-like algorithms book*. His style is to present the constituent parts of computing and from then on to build incrementally the steps required for the creation of all the well-known algorithms. This textbook is the companion for the mid-graduate course *Design and analysis of algorithms* that he teaches at the York University, Canada. Supporting slides for the course are available at http://www.cse.yorku.ca/~jeff/courses/3101/

When reading the book these slides are helpful since they illustrate how the algorithms work in a visual way and so the reader has no need to run the algorithms in her head.

The book is divided into 4 parts, iterative algorithms, recursion, optimisation problems and an appendix with some mathematical addendum aimed mainly at providing the required background for the formal evaluation of the algorithms' complexity. Each part has its own chapters but contrary to other books on algorithms, chapters are not arranged around a family of algorithms. Instead they are grouped around fundamental properties or constituent parts of each computational metaphor. Take for instance the iterative part, there are chapters about *the loop invariant in an iterative algorithm*, about types of iterative algorithms according to how they consume input or produce output, and gradually the author begins the explanation of abstract data types, search algorithms and sorting. Sure, in this book you are going to find all the algorithms you are used to in a book of its kind, but they are not arranged in a reference like fashion. They emerge as examples to support the main point of the text in the form of grayed boxes or auxiliary exercises. Here, algorithms themselves are a by-product of the theory expressed in the chapter pages and their only purpose is to illustrate the true objective, namely to learn the constituent blocks when designing algorithms. For instance, the binary search algorithm is presented when discussing the meta-heuristic *narrowing the search space* with its own properties for the loop invariant in an iterative algorithm. First we see the general concepts, and then we are shown a practical example on how to apply those concepts.

Given that this is a textbook, the main objective is that the reader should be able to master the presented material. To that end, when a subsection with new material ends you find a couple of exercises to hone the ideas. Similarly when a chapter ends, there are more exercises covering all the material in the chapter. At the end of the book, solutions to selected exercises are provided where the reader can test whether the required comprehension has been achieved. As it was said before, there are times when well-known algorithms in the literature are presented as examples in grayed boxes within the text as a complement for the proposed exercises. This emphasizes the idea that the methods are general enough and that concrete algorithms are no more than instantiations of those general concepts. The only concern about the presentation is that sometimes isn't clear where the author is aiming at and the text becomes too verbose. This is related to the fact that in almost every paragraph or two there is a text caption that introduces what comes next. This results in an impression of lack of cohesive writing that hides somehow the argumentative thread of the text.

For the functional programmer this book can be helpful in her bookshelf. Although its point of view is the imperative one, its approach can be reused in a functional environment, the programmer only has to translate the core ideas to the functional way of doing things. The abstract data types and the algorithms included in its pages are those common in a imperative environment but the main point in this book is the emphasis put in the computational aspects of the algorithms. Sure, it is difficult to translate the loop invariant concept into a functional language but the underlying idea is not to lose track of what is going on and this can also be accomplished when processing a list in a functional language. The core ideas and philosophy can be applied without any problem.

All in all this is a great book to learn how to design and create new algorithms. The author teaches you how to think about algorithms step by step, building the necessary knowledge and illustrating the process with common algorithms. The book is not the usual showcase of algorithms written in pseudo-code but anyway all the algorithms that are supposed to

be in an introductory book are in this one too. At times the prose is a little bit verbose, and the inexperienced reader can get lost because she doesn't know how the author is developing the point or what is the objective in the explanation, but the novel approach and the pedagogical freshness compensate those little deficiencies. This is a good book that the reader will appreciate in the first and subsequent reads, and it will make better developers and programmers.

TONI CEBRIÁN
cebrian@tid.es

https://doi.org/10.1017/S0956796811000177 Published online by Cambridge University Press