The Book Review Column[1]

by William Gasarch
Department of Computer Science
University of Maryland at College Park
College Park, MD, 20742
email: gasarch@cs.umd.edu

In this column we review the following books.

1. **The Art of Computer Programming Volume 4, Fascicles 2, 3, and 4** by Donald Knuth. Review by John Rogers. Volume 4 is actually out– at least in parts. I would describe it with one word: Knuthian. John Rogers, the reviewer, has more words.

2. **A Course in Computational Algebraic Number Theory** By Henri Cohen. Review by Timothy Kelley. This book provides both the algorithms and the proofs that they work. Rather comprehensive.

3. **Foundations of Computer Security** by David Salomon. Review by Richard Jankowski. The intended audience is newcomers to the field. Does it succeed? Read the review to find out!

4. **Derivation and Computation: taking the Curry-Howard correspondence seriously** by Harold Simmons. Review by Robert J. Irwin. My short description can do no better than to quote the review itself: *Simmons's text, whose thrust is pedagogical, intends to be a not-quite-one-stop-shopping experience for readers interested in the interplay between the kinds of symbol-pushing undertaken in logic and in computation.*

5. **Theoretical and Experimental DNA Computation.** by M. Amos. Review by Maulik Dave. Can we really solve problems with DNA? Read the review to find out if you should read the book to find out.

Books I want Reviewed

If you want a FREE copy of one of these books in exchange for a review, then email me at gasarchcs.umd.edu

Reviews need to be in LaTeX, LaTeX2e, or Plaintext.

Books on Algorithms and Data Structures

1. *Algorihtms on Strings* by Crochemore, Hancart, and Lecroq.

2. *Algorithms for Statistical Signal Processing* by Proakis, Rader, Ling, Nikias, Moonen, Proudler.

3. *Nonlinear Integer Programming* by Li and Sun.

4. *Binary Quadratic Forms: An Algorithmic Approach* by Buchmann and Vollmer.

5. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis* by Dey

Books on Cryptography, Coding Theory, and Security

1. **Crytanalytic Attacks on RSA** by Song Yan.
2. *Concurrent Zero-Knowledge* by Alon Rosen.
3. *Introduction to Cryptography: Principles and Applications* by Delfs and Knebl.
4. *Cryptography and Computational Number Theory* edited by Lam, Shparlinski, Wang, Xing.
5. *Coding, Cryptography, and Combinatorics* edited by Feng, Niederreiter, Xing.
6. *Formal Correctness of Security Protcols* by Bella
7. *Coding for Data and Computer Communications* by David Salomon.
8. *Block Error-Correcting Codes: A Computational Primer* by Xambo-Descamps.
9. *A consise introduction to data compression* by David Salomon.

Combinatorics Books

1. *A Coures in Enumeration* by Aigner.
2. *Computationally Oriented Matroids* by Bokowski

Auction Theory and Game Theory

1. *Putting Auction Theory to Work* by Paul Milgrom.
2. *Game Theory: Decisions, Interactions, and Evolution* by James Webb.

Logic and Verification Books

1. *Elements of Finite Model Theory* by Libkin.
2. *Finite Model Theory and its Applications* By Gradel, Kolatis, Libkin, Marx, Spencer, Vardi, Venema, Weinstein.
3. *Software Abstractions: Logic, Language, and Analysis* by Jackson.
4. *Formal Models of Communicating Systems: Languages, Automata, and Monadic Second-Order Logic* by Benedikt Bollig.
5. *Modelling Distributed Systems* by Fokkink.

Misc

1. *How to Prove it: A structured approache* by Velleman.
2. *Difference Equations: From Rabbits to Chaos* by Cull, Flahive, and Robson.
3. *Geometric Algebra for Computer Scientists: An Object Oriented Approach to Geometry* by Dorst, Fontijne, and Mann.

Review[2] of
The Art of Computer Programming
Volume 4, Fascicles 2, 3, and 4
by Donald E. Knuth
Pearson Education (Addison-Wesley), 2005
399 pages, Softcover

Review by
John D. Rogers (jrogers@cs.depaul.edu)
School of CTI, DePaul University

1 Introduction

DePaul University, where I teach, offers a Master of Science degree in Computer Science. Most of the students entering this program have very little background in CS and we help them fill this in by offering undergraduate CS courses tailored to older (and so more motivated) students.

Teaching the course in discrete mathematics is both a pleasure and a challenge. A pleasure because it is a real joy to present some of the gems of our field to people who have never encountered the mathematics we rely on. Pulling out Euclid's GCD algorithm and Euler's cycle finding algorithm and discussing the wonderful elegance of each reminds me of why I entered this field.

The challenge comes from connecting the material to practice. Many of these students come from the banking, investing, consulting, or trading firms located steps away from our downtown campus and many are working programmers. For most, the MS is a professional degree so connecting Euclid's algorithm to public key cryptography helps them understand why grasping the fundamentals of number theory or combinatorics or graph theory (and the more difficult algorithms and proofs to come in later courses) is relevant to practice.

One encounters that same pleasure in Don Knuth's latest additions to his collection of volumes titled "The Art of Computer Programming." At the same time, he also meets the challenge of practical relevance.

2 Summary

Before discussing that pleasure and challenge, let me review the scope of this multi-decade project, as described on Knuth's web page dedicated to it (`www-cs-faculty.stanford.edu/~knuth/taocp.html`). So far, he has completed the first three volumes and parts of the fourth volume of what is planned to be a seven-volume set. As many know, volume 1 deals with fundamental algorithms, volume 2 with seminumerical algorithms, and volume 3 with sorting and searching.

Volume 4 is concerned with combinatorial algorithms and will eventually appear in three (or maybe four) books. Continuing the chapter numbering of the first three volumes, this one will contain chapters 7 and 8. Volume 4A will take up sections 7.1 ("Zeroes and Ones"), 7.2 ("Generating All Possibilities"), and perhaps also 7.3 ("Shortest Paths"). The work reviewed here comprises subsection 7.2.1 ("Generating Basic Combinatorial Patterns").

Knowing that, even in retirement, it will take time to complete this opus, Knuth has decided to issue portions in *fascicles*, an old word referring to published installments of a longer work. In this case, each fascicle is a soft-covered book containing, respectively, 128, $128 + 23$, and $128 - 8$ pages. To date, there are four fascicles, each with its own table of contents and index. Fascicles 0 and 1 are scheduled to appear in 2008. Fascicle 1 of Volume 1 presents Knuth's MMIX machine and is not reviewed here.

Fascicles 2, 3, and 4 cover subsection 7.2.1, which in turn contains seven sub-subsections:

- 7.2.1.1: Generating all n-tuples
- 7.2.1.2: Generating all permutations
- 7.2.1.3: Generating all combinations
- 7.2.1.4: Generating all partitions
- 7.2.1.5: Generating all set partitions
- 7.2.1.6: Generating all trees
- 7.2.1.7: History and further references

The first two appear in fascicle 2, the next three in fascicle 3, and the last two in fascicle 4.

It may seem that this form puts a reader *in media res* (which it does!) but each fascicle has its own table of contents and index and is written so that it can stand pretty much on its own. In the text there are references to other portions of the set but most are to chapters already published and, in those cases where a reference is to something not yet available, it does not greatly impair understanding.

Knuth issued these fascicles to explain how to *generate* basic combinatorial patterns. He does not call this enumeration as this implies creating a list of objects. One usually wishes to visit the objects one by one so as to determine whether each possesses some property. Of course, being able to generate the objects one-by-one in some kind of order means being able to enumerate them. But his emphasis on generation often requires a more subtle algorithmic approach.

He moves from the simplest objects, showing how to generate all n-bit strings, to the complex, showing how to generate all n-node trees. He follows this with a 25-page history of the subject.

Most reading this review need no explanation of the objects being generated so, rather than discussing each unit, I will take as a typical example section 7.2.1.2, generating all permutations on n items.

Knuth begins by reviewing what a permutation is and then launches immediately into algorithm L (for lexicographic), which generates in lexicographic order the permutations of an ordered multiset. He traces the algorithm back to 14th century India and 18th century Germany and briefly analyzes the work L does to pass from one permutation to its successor.

Referring back to Gray code sequences in the previous (and incredibly thorough) section, where he presents algorithms that generate each n-bit string from the previous by changing only one bit, he lays out algorithm P, which generates each permutation from the previous by performing one swap of adjacent elements.

Why is this called algorithm P? That's answered in its analysis, when Knuth reveals that this is the procedure used to carry out the "plain changes" method of ringing a set of church bells. Need

a reference? There are two, pointing to 17th century English texts, followed by a brief digression into other change ringing sequences.

After a short detour into using permutation generation to solve alphametic puzzles ("`SEND + MORE = MONEY`" is the best known), Knuth launches into a general framework that makes use of the group theoretic properties of sets of permutations.

We next see algorithm G (a general permutation generator that uses a Sims table), algorithm X (lexicographic generation with restricted prefixes), algorithm H (dual permutation generator), algorithm C (generation by cyclic shifts), and algorithm E (generation using Ehrlich swaps). Each time, potentially unfamiliar terms (e.g., Sims table, Ehrlich swap) are explained and references provided.

Topological sorts come next. In this case, restrictions are placed on which order pairs of elements may appear in a permutation. Algorithm V deals with that.

Knuth ends with a caution about generating permutations, noting that one should consider whether that's the best way to solve the problem, even in the case of small instances of NP-complete problems.

To test and improve the reader's understanding he includes 112 exercises, each one scored according to its difficulty, along with solutions. The exercises are important as they often expand on some of the material. In fact, of the 69 pages devoted to this topic, 33 are taken up with exercises and solutions.

3 Opinion

Who would benefit from this book? One obvious audience is computer scientists. Speaking as one of those, I enjoyed these fascicles. Knuth is teaching me the way I try to teach my students in that introductory discrete math class, which is to convince the audience that the notion of generating combinatorial objects is intrinsically interesting AND it derives from and leads to other areas of mathematics AND it connects to pursuits outside computer science AND people from widely different times and places have wrestled with and sometimes solved associated problems AND it can be applied to solve real-world problems today. This is science writing for computer scientists.

Who else? One could use this in a course devoted to combinatorial generation. But it would also find a place in a more general algorithms course as a supplemental text. Although not written as a textbook, the presentation and the exercises give it a solid pedagogical feel. And the price is right: Each fascicle costs around US$20 so students shouldn't suffer sticker shock.

Working programmers tackling these problems will certainly welcome the diversity of approaches to solving them. I program very little but, when I need to, it's usually to test hypotheses about combinatorial objects by looking at small examples. I have found the material here very useful in generating all partially ordered sets on 10 or fewer elements in order to get an idea of the properties of poset games. I will note that Knuth relies heavily on branching in his algorithms. This allows them to be expressed succinctly and with a focus on the steps making changes to an object but it does mean that the algorithms do not leap off the page into a language like Java without rearranging some code.

The real value in this work comes from its scholarly attention to detail. Knuth recognizes and acknowledges the many and varied people and writings that provided him with algorithms, methods, variations, digressions, inspiration. Sometimes in computer science we forget that although our

discipline is hardly 60 years old we owe a debt to intellectual forebears going back many centuries. Knuth honors that debt and the text is all the richer for it.

I will end the review with a mention of the last section, the one on the history of combinatorial generation. This is good stuff. Starting with the *I Ching*, it sweeps along to the rhythmic structure of ancient Indian poetry and the metrical structure of classical Greek poetry and from there to a variety of places, objects, and people. On this sojourn he shows how authors in the past have attempted combinatorial generation in a multitude of settings and ultimately makes the point that it's not easy to get right.

But Knuth gets it right. The algorithms work, the text flows easily, the references and explanations entice, and the exercises challenge and teach. In 1999, the journal *American Scientist* named the original three volumes of *The Art of Computer Programming* one of the century's best monographs in physical science. These fascicles continue to uphold that selection.

Review of[1]
A Course in Computational Algebraic Number Theory
By Henri Cohen
Published by Springer, 2000, 534 pages
Reviewed by Timothy Kelley (klle2td@jmu.edu)
James Madison University

Introduction:

Algebraic number theory is a field of number theory that studies algebraic numbers. An algebraic number is a number that exists in the set of complex numbers and is a root of a polynomial whose coefficients are in the set of integers. That is, for a number $\alpha \in \mathbf{C}$ and polynomial $A \in \mathbf{Z}[X]$,[2] α is an algebraic number if $A(\alpha) = 0$, and A is not identically zero.[3] The computational aspect of this study involves examining, mathematically, the algorithms that solve problems related to number theory. These algorithms include fast powering algorithms, various algorithms for linear algebra, Euclid's algorithm for greatest common divisor, and elliptical curves, to name a few. These algorithms have become very important in the study of cryptography. The latest cryptography algorithms use elliptical curves, The Advanced Encryption Standard (AES) performs arithmetic over a binary finite field with a degree of 8, and RSA requires large primes to be generated and encryption and decryption require that the value of a message be raised to a large number. Thus, not only is the subject of number theory interesting in its own right, it has important real-world applications in cryptology.

Summary and Comments:

This book is divided into ten chapters and two appendices. The first six chapters are meant to be used as a full year course in the subject of computational algebraic number theory and proceed from elementary algorithms to more advanced topics such as computing integral bases in number fields. The author states two goals for his book. In the author's words, the first goal is, "to give a reasonably comprehensive introductory course in computation number theory."[4] In this aspect, I initially felt that he had failed. Upon further reflection on the subject matter and the intended audience, senior undergraduate or graduate students studying number theory, I feel that this book offers an intense look at the necessary subject matter to enter into the field of number theory. I say intense because it assumes a fairly high level of background in the subject. Luckily, the text contains a considerable bibliography with a section dedicated to essential introductory books, which contains ten books. Thus, for readers that are not familiar with the subject, but which to make themselves so, this book contains a road map to prepare for reading this book.

The second goal the author had was to write a practical guide for number theory. The book focuses not only on the fundamental algorithms but on the implementation of these algorithms, as well. When describing the algorithms in this book the author gives them in ready program form,

[2]This is the format this book uses to denote a polynomial.
[3]Henri Cohen, "A Course in Computational Algebraic Number Theory," p 153.
[4]Ibid. p XIII

gives a mathematical analysis, as well as proofs of some of the algorithms. Where proofs are not given directly they can either be found in other literature or are given as exercises for the reader. Giving the steps necessary to implement an algorithm allows the reader of this book to implement them in their favorite language and machine to get a feel for how the algorithm works. In fact, the second section of the first chapter describes the necessary components for implementing a multi-precision package, which can then be improved by adding the other algorithms found throughout the book. In this aspect, I felt the author did an excellent job. Whereas, some of the algorithms are difficult to understand in their implementation form, due to the author's choice to optimize them as well as the language used to describe them, for the most part, having the implementation aides understanding the theoretical aspects of the problem, or at the very least allows the reader to experiment with the algorithms to get a feel for them.

The first chapter contains the basic algorithms used often in number theory. This includes various powering algorithms, Euclid's algorithms, algorithms for the Chinese remainder theorem, finding primitive roots, and computing square roots modulo p. Typically the author will discus the naïve way of solving the problem and then demonstrate a faster way of doing so. For example, the naïve method for powering requires n -1 multiplications in a group. The author then describes a several more efficient methods, the simplest being the algorithm that computes:

$$g^n = \prod_{\epsilon_i=1}(g^{2^i})$$

which compute s g^n in some group G using the binary representation of n and is described as:

> **Algorithm 1.2.1** (Right-Left Binary). Given $g \in G$ and $n \in XXX[Z]$, this algorithm computes g^n in G. We write 1 for the unit element of G.
>
> 1. [Initialize] Set $y \leftarrow 1$. If $n = 0$, output y and terminate. If $n < 0$ let $N \leftarrow -n$ and $z \leftarrow g^{-1}$. Otherwise, set $y \leftarrow n$ and $z \leftarrow g$.
> 2. [Multiply?] If N is odd set $y \leftarrow z \bullet y$.
> 3. [Halve N] Set $N \leftarrow \lfloor N/2 \rfloor$. If $N = 0$, output y as the answer and terminate the algorithm. Otherwise set $z \leftarrow z \bullet z$ and go to step 2.[5]

As I mentioned above, occasionally the implementation is difficult to understand due to the way it is written. This is especially true for the more complex algorithms. After the implementation has been given the complexity typically follows as well as general discussion about interesting properties. For example, the above algorithm performs no worse than $2\lfloor lg|n| \rfloor + 1$. From this basic exploration of the fundamental algorithms, the author moves onto common problems of linear algebra, which are often necessary in number-theoretic algorithms.

Chapter two contains algorithms for solving linear algebra problems and describes the difficulties inherent in solving these types of problems, mainly stability, especially when dealing with the real numbers, which is a problem of numeric analysis. Once again, for the subjects not covered directly, the author provides several sources to the reader. The algorithms covered in this chapter include algorithms for Gaussian elimination, computing the inverse of a matrix, computing determinants, and for computing the characteristic polynomial. The second part of this chapter

[5]Ibid. p 8

deals with arbitrary $m \times n$ matrices. This part includes algorithms for finding the kernel and image of a matrix, the inverse image, supplementing a matrix (taking an $n \times k$ matrix, where $k \leq n$, and finding an invertible $n \times n$ matrix with the first k columns are the original matrix), operations on subspaces, and for computing the Hermite and Smith normal forms. This chapter also contains various algorithms for dealing with Lattices.

Chapter three begins to utilize the algorithms explored in the first chapters for operations on polynomials. Before that the author discusses the way in which they chose to represent polynomials in an actual program. This is an important discussion when trying to meet the second goal of this book, practicality. In addition to this discussion there are algorithms for finding the resultant of a polynomial, factoring polynomials, in the general sense and modulo p, and for finding the greatest common divisor between polynomials.

Chapter four begins the necessary background for algebraic number theory, the first three chapters being preparatory work for the final three chapters of the course work segment of the book. This begins more of an exploration of the theory, containing more definitions and theorems. These include the definition of algebraic numbers, as I wrote above, number fields, and the various fundamental results for both of these terms. The algorithms examined in this chapter are those for finding the signature of a polynomial, polynomial reduction, algorithms for solving the subfield problem and for field membership and isomorphism, it is important to note that these algorithms draw heavily on those algorithms described in chapters two and three, particularly the lattice algorithms and the algorithm for factoring polynomials. This chapter also lays out the primary computational tasks of algebraic number theory, which are addressed throughout the remainder of the book. These are:

1. Compute an integral basis of $\mathbf{Z}_K$, determine the decomposition of prime numbers in $\mathbf{Z}_K$, and p-adic valuations for given ideals or elements.
2. Compute the Galois group of the Galois closure of K.
3. Compute a system of fundamental units of K and/or the regulator $R(K)$...
4. Compute the class number and the structure of the class group $Cl(K)$. It is essentially impossible to do this without also computing the regulator.
5. Given an ideal of $\mathbf{Z}_K$, determine whether or not it is principal, and if it is, compute $\alpha \in K$ such that $I = \alpha\mathbf{Z}_K$.[6]

The next two chapters deal with these problems. However, chapter seven introduces, for the most part, without proofs, a survey of results about elliptic curves, and the following chapters deal with the history and future of factoring algorithms.

Here is where this book grabbed my attention. While the discussion of elliptic curves is brief, it suggests the evolving field of number theory, and as a graduate level text-book, something that is something I feel is important; not just describing what is, but what the new aspects of the field are as motivation for students wanting to go that extra bit. Certainly, the book is a bit dated, as the author states, "It is essentially impossible for an author to keep up with the rapid pace of progress in all areas of this subject."[7] The fact that elliptic curves are still cutting edge, both in

[6]Ibid. p 217
[7]Ibid. p XIII

terms of number theory and cryptography, shows that the basic principles are important, and this book serves as a good starting reference.

Chapter seven starts by defining elliptic curves and their various properties, but outlines very few algorithms. The second half of the chapter is used for describing the algorithms for working on elliptic curves. Once again, this section is brief, but like other areas that are not detailed, the author gives several books where more detailed information can be found.

The final three chapters focus on different aspects of factoring. Chapter eight describes factoring algorithms before 1970, outlining the history of factoring algorithms, as well as touching on compositeness tests, and primality tests, one of which is based on chapter seven's discussion on elliptic curves. The author makes the distinction between compositeness tests, those that determine that a number has a certain probability of being prime, and primality tests, those that prove a number is prime. Of particular interest here is Atkin's test, which utilizes the elliptic curve test and has been able to prove the primality of numbers with more than 1000 decimal places. It is also interesting to note that Atkin's test runs in polynomial time.[8] Chapter ten delves into modern factoring algorithms and eventually reaches the discussion of quadratic sieve factoring and number field sieve factoring. These are of particular interest in the field of cryptography, as many of the attacks on RSA are based on trying to factor the very large number n, which is generated by the multiplication of two large primes. So the author leaves the reader with not only a very interesting theoretical problem, but also a very relevant one.

The first appendix is an overview of available multi-precision packages for use with number theory. I will note here that since the book was first published in 1993, this discussion is a bit outdated, but the general advice is still good, as the author describes the merits and pitfalls of various commercial and freely available packages. The second appendix is a list of useful tables such as class numbers for complex quadratic fields. These are useful when experimenting with the algorithms in this book.

Conclusions:

This book is very dense. As I mentioned earlier, on first read I did not believe this could be used as an introductory text. Now I feel that it might be, but only if the students involved are well versed in both the study of number theory and the study of algorithms. This book assumes a lot of prior knowledge on the part of the reader. However, if you have the background this book can be both an excellent reference and an excellent jumping off point for further exploration. The bibliography is comprehensive and the author is careful to mention aspects of the field which he touches on lightly but are addressed in more detail elsewhere. In fact, many proofs are omitted because they can be found in other literature. While I didn?t find this book entirely useful, those sections I had background in were very interesting. Thus, my final advice is, if you have experience in the field you will find this book very useful. If you do not have the experience, but are interested in the topic, you should either, find some introductory texts and study them hard before spending the money, or purchase this book and use the bibliography as a guide for what introductory texts to purchase.

[8] Ibid. p 471

Review of[3]
Foundations of Computer Security
by David Salomon
Springer-Verlag 2006, 369 pp., $69.95, Hardcover.

Review by
Richard Jankowski jankowsr@mskcc.org

1 Introduction

The field of Computer Security probably doesn't require much of an introduction in light of recent media events. Lost and stolen laptops are placing individuals at risk of identity theft and viruses and worms are wreaking havoc across corporate networks. In *Foundations of Computer Security*, David Salomon introduces the reader to the foundational concepts of computer security. The intended audience of this book is newcomers with a limited experience with computers.

2 Summary

Salomon begins Chapter 1 with a topic that is often overlooked in security books: Physical Security. Topics such as side-channel attacks are introduced, and the author provides good coverage of the relevant and realistic threats associated with computer systems, especially portable devices. The reader will leave this chapter with the understanding that a computer is only as secure as the physical environment where it is located.

Chapter 2 is the largest chapter of the book, covering the topic of viruses. The author provides a very nice introduction into the topic covering subjects such as the motivation of virus writers, their techniques, virus propagation tactics, and classifications of the different types of viruses. An interesting part of this chapter deals with the methods viruses use to stealthily spread throughout the system. Some pseudo-code is used in examples, but all of it is of a high enough level to be approachable by the intended audience of readers.

Chapters 3 and 4 cover worms and Trojan horses respectively. Worms are introduced with a discussion of Code Red I, and an overview of the techniques worms use to spread and communicate with one another. The chapter on Trojan horses discusses the applications of Trojan horses and techniques of installing Trojan applications on systems. The concept of rigging a compiler is covered, but the code used in the examples may be a little over the heads of the target audience of this book (newcomers with only a basic knowledge of computers).

Chapter 5, Examples of Malware, is a nice chapter that covers the history of a dozen of the more well-known viruses and worms. Most readers will probably find this chapter interesting, as they can compare mainstream viruses with one another. Household-name viruses, like Michelangelo, SirCAM, and Melissa are all discussed.

Chapter 6, Prevention and Defenses, deals with strategies to defend against the malicious software described in the four previous chapters. The author discusses anti-virus software, backups, and closes with a section on educating users about virus hoaxes.

Network Security is covered in Chapter 7, and the author does a very good job at introducing the essentials. Port scanning and the various forms of spoofing are first introduced. The author

then dedicates several pages to spam covering topics such as spam proxies, filtering, and how to avoid spam from a user perspective. Sound advice is given here, such as turning off preview mode in your mail reader and not responding to any spam messages. Salomon closes this chapter with an introduction to Denial of Service and Firewall Basics.

Chapter 8 deals with Authentication, and after a short introduction to what authentication is, the author discusses common biometric authentication systems, such as fingerprints and facial recognition. The author closes the chapter with a very complete and important chapter on passwords. The reader is given recommendations on developing good passwords, and threats to passwords, such as dictionary attacks and sniffing are discussed.

Spyware is covered in Chapter 9. This author does a good job discussing the political, economical, and other motivations behind the deployment of spyware. How spyware is distributed to systems is discussed, as well as the differences between similar applications such as adware.

Chapter 10 covers identity theft, a very popular and timely topic. The author covers best practices for avoiding being victimized, such as monitoring your credit and shredding unneeded documents. Sound advice is given to destroy magnetic and optical storage when disposing of them, and phishing techniques are discussed. The author closes this chapter with a technique called a homograph threat, where the reader is introduced to the concept that the Web site they are visiting may, in fact, be a spoofed site.

Privacy and Trust is the title for Chapter 11, and the author gives the reader practical advice to protect themselves and their children. Trust is introduced in the form of consumer trust in making online purchases.

Chapter 12, the final chapter of the book, introduces the reader to the basic elements of cryptography. Basic ideas are introduced, such as the importance of keeping the encryption key secret, monoalphabetic and polyalphabetic ciphers, one-time pads, and public key cryptography. It's a lot to squeeze into the 21 pages of the chapter, and like in Chapter 4, the author may start to go over the heads of the intended audience, especially in the RSA section. The chapter closes with SSL and a nice introduction to how the protocol is employed.

In addition to the main chapters, appendices cover the language notation system used by the hacker underground and a timeline of viruses up until 2005. Also included is a "Concluding Remarks" section that distills much of the advice given throughout the book in a way that the reader can use as a take-away list.

3 Opinion

Overall, *Foundations of Computer Security* is a very nice and well-written introduction to the essential concepts of computer security. The content was written in an easy tone that would make it approachable to a security neophyte. The book is loaded with examples and exercises that are useful in learning the material.

One of the things I found appealing about the book is much of the content is written in bulleted lists. In my opinion, this allows the reader to quickly get an understanding of the core concepts of the material.

This is a book I would not hesitate to give to someone who had very little computer experience and wanted to learn the core concepts of security. The material covered is very broad in scope; however the essentials were well treated and easy to comprehend.

Review[4] of
Derivation and Computation: taking the Curry-Howard correspondence seriously
Author of book: Harold Simmons
Series: Cambridge Tracts in Theoretical Computer Science, Volume 51
Cambridge University Press, 2000, Hardcover, 384 pages, $75.00

Review by
Robert J. Irwin, Hamilton College

1 Overview

The term *Curry-Howard correspondence* (also: *Curry-Howard isomorphism* or *formulae-as-types correspondence*) refers to the relationship between formal calculi for logical derivations and formal calculi for computations. For example, minimal logic — the implicational fragment of intuitionistic propositional logic — is isomorphic to the simply-typed λ-calculus. How so? If we associate the propositional variables of the logic with the type variables of the λ-calculus, the logical formulae will then correspond to simple types. Moreover, proofs in minimal logic correspond to λ-terms of appropriate type, and the provability of a formula corresponds to the existence of a λ-term of that type (i.e., to that type's being *inhabited*). The germinal idea, due to Haskell B. Curry [1, 2] in the context of combinatory logic with type assignment, was applied to typed λ-calculi and popularized by William A. Howard in a manuscript widely circulated since 1969, but only published in 1980 [3].

The subtitle is a bit curious — folks have "[taken] the Curry-Howard correspondence seriously" for decades and many texts on type theory feature the correspondence. For example, the influential and oft-cited book that inspired the author to write the one under review [4] is about nothing else, and the excellent introductory text [7] devotes a chapter to the subject. Readily available lecture notes [5] treat the correspondence, and no modern textbook on the theory of programming languages (e.g, [6]) can be complete without at least mention of the correspondence.

While the correspondence between minimal logic and simply-typed λ-calculus is particularly transparent, it turns out that an odd assortment of background knowledge is needed to progress to isomorphisms between richer pairs of logical and computational calculi. Simmons's text, whose thrust is pedagogical, intends to be a not-quite-one-stop-shopping experience for readers interested in the interplay between the kinds of symbol-pushing undertaken in logic and in computation.

2 Summary of Contents

As the author writes in the Introduction, "[t]here is nothing worse than an exercise you can't do and have no way of finding a solution to it." That is why, following about 200 pages of exposition, with exercises, the remainder of the text is devoted to the presentation of (mostly) fully worked solutions to *all* the preceding exercises, of which there are approximately 200.

After the Introduction comes a "Preview" chapter, which will help readers integrate the tuition they are about to receive in the next nine chapters. Only a background in basic logic is assumed — Simmons does not assume other authors have conditioned his readers for the news. Chapters on "Derivation Systems" (minimal logic via natural deduction and Hilbert-type formalisms), basic "Computation Mechanisms" (untyped combinatory logic and λ-calculus), "The Typed Combinator Calculus", "The Typed λ-Calculus" and "Substitution Algorithms" prepare the way for the key

chapter on "Applied λ-Calculi." In the last-mentioned chapter, the emphasis shifts to the interpretation of types as function spaces and terms as particular functions, leading to the investigation of recursion and induction over the natural numbers and Gödel's sytem T [9] (the primitive recursive functionals).

Subsequent chapters cover "Multi-Recursive Arithmetic", "Ordinals and Ordinal Notations" and, finally, "Higher Order Recursion". The multi-recursive arithmetic chapter studies recursion in some detail, covering simultaneous recursion over multiple arguments, etc., the sort of material typically given short shift in introductory (and many advanced) treatments of recursion theory, but featured in more specialized works like [10, 11]. Higher-order formulations of recursion are captured via the calculus $\lambda\mathbf{G}$, essentially Gödel's T minus the equational reasoning. Hierarchies of number-theoretic functions are studied and their relative complexities are shown to correspond with their formulations in $\lambda\mathbf{G}$.

The chapter on ordinals and ordinal notations provides material often omitted from less self-contained treatments. Still, as in other chapters, not all the necessary background is provided for doing all the related exercises, though the reader is alerted when extra information is needed, and given references to the select bibliography to supply the missing pieces (e.g., a proof of the Cantor normal form theorem). The significance of the differences between ordinals and ordinal notations is clearly established. Treatment of the ordinals less than the first critical ordinal ϵ_0 is given, which allows the analysis of $\lambda\mathbf{G}$ to be completed in the final expository chapter.

Following the exposition are nine more chapters, each containing solutions to the exercises of the corresponding expository chapter. Solutions are presented with a level of care and detail seldom encountered in other texts.

3 Opinion

This text is recommended for the student or researcher who's been exposed to bits and pieces of the Curry-Howard correspondence, but wants a sharper idea of the big picture and is willing to work through the exercises to see how the details fit together. Simmons has succeeded in pulling together the main fruits of the correspondence for simple types in a single text.

Regarding style, the book is rather snappily written. It's informal, breezy — sometimes positively jaunty — and always directly addressed to the reader. Rarely, the informality partly defeats the explanation of parts of what is, essentially, a highly formal subject. Though well thought-out overall, and elaborately typeset, the book retains some of the feel of lecture notes, from which it in fact evolved. For example, the occasional definition is oddly written, with the definiendum unclear or withheld to the last.

The short, but annotated and well-selected bibliography is sufficient to fill the gaps in the text. Still, I wouldn't recommend the book for rank beginners, who would be well-prepared by learning the material in [7], which treats almost exclusively Curry's simple type theory ("type-assignment"), and perhaps a good, compatible proof theory book, say [8]. These two texts combined contain no more exercises than Simmons's, and only a fraction of them are solved.

From the Preview: "I could rationalize the choice of topics, but in the end this wouldn't convince you if I have missed your favourite." What I miss is Schwichtenberg's result that the "extended" polynomials (polynomials plus the conditional) are precisely the functions definable in the simply-typed λ-calculus [12], which could have been presented nicely in a few short exercises.

It can't be emphasized enough that the great thing about this book is its many well-chosen, completely solved exercises. This alone makes it a valuable text, especially for self-study.

References

[1] H.B. Curry. Functionality in Combinatory Logic, *Proceedings of the National Academy of the U.S.A.*, 20 (1934), 584-590.

[2] H.B. Curry and R. Feys. *Combinatory Logic, Vol. 1*, North Holland, 1958; 2nd ed., 1968.

[3] W.A. Howard. The formulae-as-types notion of construction, in *To H.B. Curry*, ed. J.R. Hindley and J.P. Seldin, Academic Press, UK 1980, pp. 479-490.

[4] J-Y. Girard, Y. Lafont and P. Taylor. *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science, Volume 7, Cambridge U. Press, 1989.

[5] M.H. Sorensen and P. Urzyczyn. *Lectures on the Curry-Howard isomorphism*. Technical Report 98/14, DIKU, Copenhagen, 1998.

[6] B.C. Peirce. *Types and Programming Languages*, MIT Press, 2002.

[7] J. Roger Hindley. *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science, Volume 42, Cambridge U. Press, 1997.

[8] A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory, 2nd ed.*, Cambridge Tracts in Theoretical Computer Science, Volume 43, Cambridge U. Press, 2000.

[9] K. Gödel. Über eine bisher noch nicht unbenützte Erweiterung des finiten Standpunktes, *Dialectica* 12:280-287, 1958.

[10] R. Péter. *Recursive Functions*. Academic Press, 1967.

[11] H.E. Rose. *Sub-Recursion: Functions and Heirarchies*, Oxford Logic Guides, Oxford U. Press, 1984.

[12] H. Schwichtenberg. Definierbare Funktionen im λ-Kalkül mit Typen, *Archiv für Mathematische Logik und Grundlagenforschung*, 17:113-114, 1975.

Review of
Theoretical and Experimental DNA Computation. [5]
by M. Amos
Published in 2005 by Springer-Verlag Berlin Heidelberg, 172 pages

Reviewer: Maulik A. Dave

1 Overview

Interest in DNA computing has increased among research scientists since 1990s. Author himself is a scientist working in DNA computing. The book is an overview of DNA computing. It touches both theoretical and experimental aspects. The theoretical aspects include algorithms, and computing models. The experimental aspects consist of various DNA experiments done for computing.

2 Content Summary

The book is divided into 6 chapters.

2.1 DNA molecule and operations on DNA

The first chapter gives a general introduction to DNA. A brief explanation of the double helix structure of DNA is followed by a detailed explanation of operations on DNA. The terms such as synthesis, denaturing, and ligation are explained. Cloning of DNA is described in details. Techniques such as Gel Electrophoresis for sorting DNA by size, Restriction Enzymes for recognizing specific DNA, and Polymerase Chain Reaction for amount of DNA in a given solution, are described.

2.2 Theoretical Computer Science

The second chapter introduces theoretical computer science. It begins with a brief section on gates, and boolean algebra. Models such as automata, Turing machine, and Random Access machine are described. Algorithms, and data structures are introduced. An introduction to computational complexity theory includes the NP completeness.

2.3 DNA Computational Models

The models are referred as molecular because they described at abstract level. The models are classified into filtering models, splicing models, constructive models, and membrane models. In filtering models, a computation consists of a sequence of operations on finite multi-sets of strings. With other models, a filtering model made by author, called parallel filtering model, is described in details. The parallel filtering model has remove, union, copy and select operations. NP algorithms written using these operations are also described. Splicing models has a splicing operation, which takes two strings, and concatenates prefix, suffix of one another. Constructive models utilize principal of self assembly. Models, which are chemical, or biological, but not DNA based are described under the term membrane models.

2.4 Complexity of algorithms in DNA Computing

The chapter on complexity presents more insight into the DNA computing. It discusses in details, two boolean circuit models. A DNA computing model, which is able to simulate boolean circuits, is a Turing-complete model. The discussion on such models include introduction to the model, laboratory implementation, an example application, and complexity analysis. Later in the chapter, P-RAM simulation on DNA computing is discussed. Processor instruction set, overview of underlying controller, and translation process from P-RAM algorithms to DNA implementation are described. Further, the P-RAM model is explained by using List ranking problem as an example.

2.5 Laboratory experiments

Laboratory implementations of the algorithms are described with their actual experimental details in last two chapters. Author's own work is described in more details. It is compared with Alderman's work. The chapters do not confine to the DNA computing only. It describes implementation of Chess games on RNA. Groups of organisms having two nuclei, and possessing hair like cilia for movement are called Ciliates. A small chapter is devoted for describing computing based on ciliates.

3 Conclusion, and Comments

A high school level knowledge of biology, particularly of DNA is sufficient to understand the experimental descriptions in the book. However, to appreciate the computing part of DNA computing, knowledge of computers at undergraduate level is necessary. The book is a good introduction to DNA computing for both new researchers, and readers having general interests.