

**The Book Review Column**<sup>1</sup>  
by William Gasarch  
Department of Computer Science  
University of Maryland at College Park  
College Park, MD, 20742  
email: [gasarch@cs.umd.edu](mailto:gasarch@cs.umd.edu)

In this column we review the following books.

1. **Modern Computer Algebra** by Joachim von zur Gathen and Jürgen Gerhard. Reviewed by R. Gregory Taylor. This is a high level book on algorithms that do algebraic operations on polynomials and other objects.
2. **The Discrepancy Method — Randomness and Complexity** by Bernard Chazelle. Reviewed by Jin-Yi Cai. This is a high level book on complexity theory as it interacts with randomness.
3. Joint review of **Computability and Complexity Theory** by Steven Homer and Alan L. Selman, and **The Complexity Theory Companion** by Lane A. Hemaspaandra and Mitsunori Ogihara. Reviewed by E. W. Čenek. Both of these are books on complexity theory that could be used in a graduate course.
4. **Mathematical Theory of Domains** by Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor. Reviewed by P. Daniel Hestand. Domains are complete partial orders with least element. They are used in denotational semantics; however, this is the first full book about them.

**I am looking for reviewers for the following books**

If you want a FREE copy of one of these books in exchange for a review, then email me at [gasarch@cs.umd.edu](mailto:gasarch@cs.umd.edu)

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

**Books on Algorithms, Combinatorics, and Related Fields**

1. *Flexible pattern matching in springs (practical on-line algorithms for texts and biological sequences)* by Navarro and Rarrinot.
2. *Diophantine Equations and Power Integral Bases* by Gaal.
3. *Algorithm Design: Foundations, Analysis, and Internet Examples* by Goodrich and Tamassia.
4. *An Introduction to Data Structures and Algorithms* by Storer.
5. *Structured Matrices and Polynomials: Unified Superfast Algorithms* by Pan.
6. *Computational Line Geometry* by Pottmann and Wallner.
7. *Algorithms Sequential and Parallel* by Miller and Boxer.
8. *Computer Algorithms: Introduction to Design and Analysis* by Basse and Gelder.

---

<sup>1</sup>© William Gasarch, 2002.

9. *Linear Optimization and Extensions: Problems and Solutions* by Alevras and Padberg.
10. *An Introduction to Quantum Computing Algorithms* by Pittenger.

### **Books on Cryptography and Books on Learning**

1. *Introduction to Cryptography* by Buchmann.
2. *Elliptic Curves in Cryptography* by Blake, Seroussi, and Smart.
3. *Coding Theory and Cryptograph: The Essentials* by Hankerson, Hoffman, Lenoard, Linder, Phelps, Rodger, and Wall.
4. *Learning with Kernels (Support Vector Machines, Regularization, Optimization, and Beyond)* Bernard Scholkopf and Alexander Smola.
5. *Learning Kernel Classifiers* by Herbrich.

### **Books on Complexity and Logic**

1. *Models of Computation: Exploring the Power of Computing* by John Savage.
2. *Complexity and Information* by Traub and Werschulz.
3. *Derivation and Computation* by Simmons.
4. *Types and Programming Languages* by Pierce.
5. *Logic and Language Models for Computer Science* by Hamburger and Richards.

Review of

#### **Modern Computer Algebra**

**Authors: Joachim von zur Gathen and Jürgen Gerhard**

**Publisher: Cambridge University Press**

**Hardcover, xiii + 753 pages**

Reviewer: R. Gregory Taylor<sup>2</sup>

Department of Computer Science

Trinity College (Hartford, Connecticut USA)

## **1 Overview**

Modern computer algebra is characterizable as that branch of computer science wherein mathematical techniques and allied software are directed toward obtaining exact solutions for problems in science and engineering. One form the search for such solutions can take is the search for exact solutions to some sort of equation or inequality. (In contrast, numerical analysis concerns itself with approximate solutions.) In any case, this is what authors Von zur Gathen and Gerhard mean by “computer algebra,” and this conception guides their selection of topics for the remarkable book under review. Of course, computer algebra can also encompass the application of group-theoretic ideas in the design of software for generating or enumerating configurations of various sorts and

---

<sup>2</sup>©R. Gregory Taylor 2002

for determining whether two representations are equivalent in some important way. However, this combinatorial branch of the subject is not covered in this book.

The focus of most of the book is polynomials. (The exception here is Part IV on primality.) Thus an algorithm whose input consists of one or more polynomials is routinely described first. Only afterward is the corresponding algorithm with input consisting of (multiprecision) integers handled as a special case. So, for example, in §2.3 the classical algorithm for multiplying two single-variable polynomials  $\sum_{0 \leq i \leq n} a_i x^i$  and  $\sum_{0 \leq i \leq m} b_i x^i$  is described and analyzed as involving  $O(n \cdot m)$  arithmetic (ring) operations. Subsequently, multiplication of two multiprecision integers  $(-1)^s \times \sum_{0 \leq i \leq n} a_i 2^{64i}$  and  $(-1)^t \times \sum_{0 \leq i \leq m} b_i 2^{64i}$  with  $s, t \in \{0, 1\}$  is quickly described and the cost analysis of  $\tilde{O}(n \cdot m)$  word operations mentioned. (A 64-bit word is assumed.)

In giving the cost analysis of the polynomial, but not the integer, version of an algorithm, the authors always provide an explicit constant for the dominant term. The authors use so-called *soft- $O$  notation* to hide ugly logarithmic factors. Technically, function  $f(n)$  is  $O^\sim(g(n))$  if  $f(n)$ , for sufficiently large  $n$ , is bounded above by  $g(n)$  times a polynomial in the logarithm of  $g(n)$ . Thus function  $n \cdot \log n \cdot \log \log n$  is  $O^\sim(n)$ .

**Modern Computer Algebra** is divided into five parts, and each part is associated with a historical figure whose ideas have been especially influential.

Part I (Euclid). Chapters 2–7. The Euclidean Algorithm. Modular Algorithms and Interpolation. The Resultant and GCD Computation.

Part II (Newton). Chapters 8–13. Fast Multiplication. Newton Iteration. Fast Polynomial Evaluation and Interpolation. The Fast Euclidean Algorithm. Fast Linear Algebra.

Part III (Gauß). Chapters 14–17. Factoring Polynomials over Finite Fields. Hensel Lifting. Short Vectors in Lattices.

Part IV (Fermat). Chapters 18–20. Primality Testing. Factoring Integers.

Part V (Hilbert). Chapter 21–24. Gröbner Bases. Symbolic Integration and Summation.

Appendix Mathematical Prerequisites.

Each of the five parts opens with a full-color plate—several of them quite stunning visually—and a very short, but informative, biographical sketch of its leading figure. We cannot resist quoting Von zur Gathen and Gerhard on the life of Hilbert.

Hilbert liked lecturing, and excelled at it. He usually prepared only an outline of his lecture and filled in the details in front of the students—so he got stuck and confused everybody at times, but “a third of his lectures were superb.” (p. 561)

## 2 Summary of Contents

Chapter 1 previews three application areas that figure in later parts of the text: cyclohexane ( $C_6H_{12}$ ) conformations, the RSA cryptosystem, and secret sharing (distributed data structures). The authors’ explanation of the RSA system is brief and easy to follow. Since these particular application areas arise only in specific chapters appearing toward the ends of Parts V, IV, and I, respectively, the authors’ intention, here in Chapter 1, is only to motivate the reader: by sampling the areas to which their algorithms can be applied, they make the point that these algorithms do have impressive and, in some cases, surprising applications.

## 2.1 Part I (Euclid)

In Chapter 3, all the important properties of the Extended Euclidean Algorithm (EEA), in particular, the result that  $\gcd(a, b) = sa + tb$ , where  $s$  and  $t$  are the so-called *Bézout coefficients* of  $a$  and  $b$ , are proved in full generality, i.e., for  $a$  and  $b$  elements of an arbitrary Euclidean domain (integral domain having a certain division property). The proofs proceed by way of elegant arguments involving matrices. Typically, textbook authors emphasize the usefulness of  $\gcd(a, b)$ ,  $s$ , and  $t$  only. But Von zur Gathen and Gerhard show that the intermediate results computed by the EEA are also not without their uses. To take a simple example, in §4.6 they show that the sequence of quotients generated by EEA, applied to integers  $a$  and  $b$ , gives the continued fraction expansion of  $\frac{a}{b}$ .

As an example of their policy of providing explicit constants in time analyses, the authors' Corollary 4.6 states that, where  $F$  is a field and  $f$  is a polynomial in  $F[x]$ , one multiplication modulo  $f$  in the residue class ring  $F[x]/\langle f \rangle$  involves  $4n^2 + O(n)$  arithmetic operations in  $F$ , worst case, and one inversion modulo  $f$  involves  $\frac{13}{2}n^2 + O(n)$  operations in  $F$ . On the other hand, no such constant is provided in Corollary 4.7, where operations on multiprecision integers are the topic.

The provided cost analysis of EEA leads to a brief and interesting digression into the probability that a randomly chosen pair of natural numbers is relatively prime. (This turns out to be  $\approx \frac{6}{\pi}$ .) The discussion incorporates a software-generated image (Figure 3.2) of the frequency of relative primality. This is just one of many such graphic displays in the book, some of them multicolored and strikingly beautiful.

A number of applications of the EEA are covered in some detail: modular arithmetic, linear Diophantine equations, and continued fractions. The latter in turn leads to delightful and instructive digressions into the design of astronomical calendars and musical scale systems. We learn why subdivision of the octave into twelve half-tone steps was a good choice, based on the theory of diophantine approximation of arbitrary reals by sequences of continued fractions. (Another good choice would have been nineteen third-tone steps.) The last of 32 exercises in Chapter 4 relates Sturm's Theorem to the classical Euclidean Algorithm (no Bézout coefficients).

Chapter 5 introduces three variants of modular algorithms featuring primes in  $Z$  or, more generally, in any Euclidean domain: these variants involve either a single large prime, a collection of small primes, or a single prime power. The principal application domain is polynomial interpolation, and the Chinese Remainder Algorithm is the main vehicle in the case of integers. Applications to partial function decomposition are also discussed.

Chapter 5 focuses upon applications of the Chinese Remainder Algorithm, involving calls to EEA, in the context of recovery of the coefficients of a polynomial from its values at several points. Such *interpolation* is introduced by way of Lagrange interpolants. (For its inverse, namely, polynomial evaluation, there is Horner's Rule.) The former idea has application to secret sharing: a secret is distributed over the  $n$  members of a group of individuals, and the goal is that, together, they can make use of it but no proper subset of the group can do so—a situation that might arise in the case of a shared bank account. The Chinese Remainder Algorithm is introduced (Algorithm 5.4), and Chinese remaindering is then applied to integers as well as polynomials. The reader is encouraged to regard the Chinese Remainder Algorithm as a generalization of Lagrange's idea.

The authors show how the determinant of a matrix  $A_{n \times n}$  with integer coefficients can be computed using a collection of small prime moduli and that the number of word operations involved, worst case, is  $O(n^4 \log_2 B)$ , where  $|a_{ij}| \leq B$  for  $1 \leq i, j \leq n$  (Theorem 5.12). Section 5.6 presents an application of Chinese remaindering to Hermite interpolation. Section 5.7 covers the problem of finding a rational function  $r/t \in \mathbf{F}[x]$ , for some field  $\mathbf{F}$ , that is of "small degree" and that is

congruent to some given polynomial modulo another polynomial.

Chapter 7 shows how to decode BCH codes by applying the theory of Padé approximants and the Extended Euclidean Algorithm.

## 2.2 Part II (Newton)

This part of the book consists largely of the description of fast, i.e., almost linear-time, algorithms for problems in polynomial algebra for which classical, quadratic-time algorithms were given in Chapters 2–5 of Part I: multiplication, division with remainder, modular multiplication, radix conversion, multipoint evaluation, interpolation, reduction modulo several moduli, determination of the greatest common divisor, and modular inversion. Fast multiplication is essential for all of them.

First, Karatsuba’s algorithm for multiplying two polynomials of degree less than  $2^n$  is described, and it is shown that it involves  $O(n^{1.59})$  additions and multiplications. Next, the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) are described, and it is shown that two polynomials  $f$  and  $g$  with  $\deg(fg) < n$  can be multiplied in  $\frac{9}{2}n \log n + O(n)$  additions and multiplications. Arithmetic circuits are given for both Karatsuba and FFT, as are graphical representations of computation costs. Finally, Schönhage and Strassen’s algorithm for multiplying polynomials of degree less than  $n = 2^k$ , for some  $k$ , is shown to compute using  $\frac{9}{2}n \log n \log \log n + O(n \log n)$  ring operations.

The ring-theoretic descriptions of DFT and FFT, as presented in §8.2, are particularly concise. A sequence of examples and lemmas lead to Theorem 8.13 stating that, where  $\omega$  is a primitive  $n$ th root of unity in the context of ring  $R$ , Vandermonde matrix  $V_\omega$  of  $DFT_\omega : f \mapsto \langle f(1), f(\omega), \dots, f(\omega^{n-1}) \rangle$  is invertible with inverse  $\frac{1}{n}V_{\omega^{-1}}$ . The FFT of Cooley and Tukey can then be used to compute the DFT in  $O(n \log n)$  operations in  $R$ , which, under the right circumstances, in turn gives an  $O(n \log n)$  multiplication algorithm for polynomials in  $R[x]$  of degree less than  $n$  (Corollary 8.19). An instructive arithmetic circuit illustrating the FFT for the case  $n = 8$  is provided. Chapter 13 introduces the Continuous Fourier Transform and shows its application to image processing (audio and video compression).

Subsequent to Chapter 8, whenever some routine for polynomial multiplication must be called, the time analysis of the calling algorithm involves a multiplication-time parameter  $\mathcal{M}(n)$  expressing the cost, worst case, of multiplying using either the classical algorithm or either of two fast algorithms (Karatsuba or Schönhage–Strassen).

Section 9.7, entitled “Implementations of Fast Arithmetic,” is a nine-page discussion with diagrams, of two software packages: Victor Shoup’s NTL and the authors’ own BIPOLAR. By coding and testing competing algorithms, one first identifies *crossover points*—input sizes whereby one algorithm begins to outperform another. Subsequent implementations are then *hybrid* to the extent that an algorithm “kicks in” just in case input size lies above some one crossover point (but perhaps below another). The discussion illustrates this nicely for the classical algorithm for polynomial multiplication, Karatsuba’s algorithm, and an algorithm of Cantor. Figure 9.5 identifies the two crossover points involved.

Section 10.1 describes a fast algorithm for multipoint evaluation, i.e., for evaluating a polynomial  $f \in \mathbf{R}[x]$  with  $\deg(f) < n$  at points  $u_0, \dots, u_{n-1}$ . As a preliminary, all subproducts in a certain binary subproduct tree for  $u_0, \dots, u_{n-1}$ , of height  $\log_2 n$  are computed. Afterward, this subproduct tree is used to compute  $f(u_0), \dots, f(u_{n-1})$  by recursively computing  $r_0(u_0), \dots, r_0(u_{\frac{n}{2}-1})$  and  $r_1(u_{\frac{n}{2}}), \dots, r_1(u_{n-1})$  for related polynomials  $r_0$  and  $r_1$ . The algorithm computes in  $O(\mathcal{M}(n) \log n)$  ring operations. A proof of correctness is provided. .

### 2.3 Part III (Gauß)

Part III opens with a discussion of algorithms for factorization of single-variable polynomials over finite fields (Chapter 14). A typical algorithm proceeds in three stages: (1) “square-free” factorization isolating duplicated factors; (2) “distinct-degree” factorization yielding factors no two of which have the same degree; and (3) “equal-degree” factorization in turn decomposing the factors in (2) into factors having the same degree. Step 3 here is the most involved, and the authors present a probabilistic algorithm of Cantor and Zassenhaus for finding a *splitting polynomial* for input polynomial  $f$  of degree  $n > 0$  over a finite field of order  $q$  such that all irreducible factors of  $f$  are known to have degree  $d$ . A polynomial  $a$  with degree less than  $n$  is chosen at random and verified to satisfy  $\gcd(a, f) = 1$ . Then, for a certain  $e(n, d)$ , polynomial  $a^e - 1$  is computed and in effect checked to see whether it splits  $f$ . If so, then a single factor of  $f$  is returned. The probability that, for the chosen  $a$ , polynomial  $a^e - 1$  fails to split  $f$  is less than or equal to  $\frac{1}{2}$ . This probability can be made arbitrarily small by running the algorithm repeatedly. The presentation of the Cantor–Zassenhaus algorithm includes a multicolored diagram (Figure 14.5) illustrating the situation with respect to a lucky or unlucky choice for  $a$ . An expected-case time analysis is provided in Theorem 4.11.

Chapter 15 presents modular algorithms for factoring in  $\mathbf{Q}[x]$  and in  $\mathbf{F}[x, y]$  for field  $\mathbf{F}$ . Chapter 16 describes the so-called *LLL algorithm*, due to Lenstra, Lenstra, and Lovász, for factoring in  $\mathbf{Z}[x]$ . It computes in time polynomial in the degree of input polynomial  $f$ .

### 2.4 Part IV (Fermat)

This part is somewhat shorter than the others. It reviews investigations concerning prime numbers, i.e., primality testing and the search for prime numbers. A starting point is Fermat’s Little Theorem stating that if  $N$  is prime and  $\gcd(a, N) = 1$ , then  $a^{N-1} \equiv 1 \pmod{N}$ . It follows that if  $N$  is prime and  $a < N$ , then  $a^{N-1} \equiv 1 \pmod{N}$ . It turns out, on the other hand, that there exist composite numbers  $N$ —the so-called *Carmichael numbers*, of which there are infinitely many—that have this property as well.

The Fermat Primality Test for odd input  $N$  amounts to randomly choosing some even  $a$  less than  $N$  and then testing  $a$  and  $N$  for the Little Fermat property with a cost of  $O(\log N \cdot \mathcal{M}(n))$  word operations. (Again,  $\mathcal{M}(n)$  is a multiplication-time parameter—this time for integer multiplication.) If  $a$  and  $N$  are seen to lack the property (output “ $N$  is composite”), then, by Little Fermat,  $N$  is composite without question, although the algorithm supplies no factor. But if  $a$  and  $N$  are seen to possess the property (output “ $N$  is probably prime”), then one can speak only in probabilities. The result may reflect the fact that  $N$  really is prime, and any other choice of even  $a < N$  would have given the same result,  $N$  having no *Fermat witnesses*. (But since we have not looked at all those other choices, we cannot know this.) However, the same will be true if  $N$  is (composite and) Carmichael. Finally, if  $N$  is composite but not Carmichael, then the result reflects only bad luck: our  $a$  is a *Fermat liar* and some other choice of  $a$  would have witnessed  $N$ ’s compositeness. And this really is bad luck, since, by Lagrange, a composite  $N$  that is not Carmichael has at least as many Fermat witnesses as Fermat liars. In other words, if  $N$  is composite and not Carmichael, the Fermat test will give the correct answer “ $N$  is composite” with a probability of at least  $\frac{1}{2}$ .

It is a fact that integer  $N$  is Carmichael if and only if (1)  $N$  is square-free and (2) any prime factor  $p$  of  $N$  is such that  $p - 1$  is a factor of  $N - 1$ . Also, any Carmichael number is odd and has at least three (distinct) prime factors. This has led to a refinement of the Fermat test, discussed by von zur Gathen and Gerhard in §18.3. Again, for odd input  $N$ , even  $a < N$  is chosen at random. If  $g = \gcd(a, N) > 1$ , then factor  $g$  of  $N$  is returned. Otherwise, by  $N$  odd, we can find  $k \geq 1$  and odd  $m$  such that  $N - 1 = 2^k m$ . We compute  $b_0 = a^m \pmod{N}$ . If  $b_0 = 1$ , the algorithm returns

“ $N$  is probably prime.” If  $b_0 \neq 1$ , then the assignment  $b_i = b_{i-1}^2 \bmod N$  is iterated  $k$  times to give  $b_k$ . Now if  $b_k \neq 1$ , then the algorithm returns “ $N$  is composite.” Otherwise,  $j$  becomes the least  $i$  with  $b_{i+1} = 1$ . (There must be such since  $k$  itself is such a  $j$ .) Finally,  $g = \gcd(b_j + 1, N)$  is computed. If either  $g = 1$  or  $g = N$ , then “ $N$  is prime” is returned and otherwise, factor  $g$  is returned. The improvement over the Fermat test resides in the fact that if input  $N$  happens to be Carmichael, then the algorithm more likely than not returns a factor of  $N$ . The running time is that of the Fermat test, i.e.,  $O(\log N \cdot \mathcal{M}(N))$ . Repeated execution can be used to force the error probability below any desired limit, and the authors summarize their discussion of this “Strong Pseudoprimality Test” as follows.

What does it mean when a primality test returns “probably prime” on input  $N$ ? Is  $N$  then “probably prime”? Of course not;  $N$  is either prime or it is not. If we have run the test 1001 times, say, then it means the following: if  $N$  is not prime, then an event has been witnessed whose probability is at most  $2^{-1001}$ . If you fly in an airplane whose safety depends on the actual primality of such an “industrial-strength pseudo-prime”, then this fact should not worry you unduly, since other things are much more likely to fail ;-) (page 497)

Section 18.4 describes probabilistic algorithms for finding primes, all of them based upon the Prime Number Theorem from nineteenth-century analytic number theory stating that the number  $\pi(x)$  of primes  $\leq$  real  $x$  is approximately  $\frac{x}{\ln x}$  and that the  $n$ th prime  $p_n$  is approximately  $n \cdot \ln n$ . If a large prime between  $B$  and  $2B$  is sought—there must be one by Bernard’s Postulate—then we randomly select numbers  $p$  with  $B < p \leq 2B$  and test them for primality using Strong Pseudoprimality until we find one that passes  $k$  tests for a fixed  $k$ . The result of this procedure is prime with probability not less than  $1 - 2^{-k+1}$ , and the expected cost is  $O(k(\log^2 B)\mathcal{M}(\log B))$  word operations.

The short §18.5 quickly reviews the probabilistic polynomial-time algorithm for primality testing due to Solovay and Strassen. Where  $N$  is prime and  $\left(\frac{a}{N}\right)$  is the Legendre symbol defined by

$$\left(\frac{a}{N}\right) = \begin{cases} 1 & \text{if } \gcd(a, N) = 1 \text{ and } a \text{ is a square modulo } N \\ -1 & \text{if } \gcd(a, N) = 1 \text{ and } a \text{ is not a square modulo } N \\ 0 & \text{if } \gcd(a, N) \neq 1 \end{cases}$$

we have that  $\left(\frac{a}{N}\right) \equiv a^{\frac{N-1}{2}} \bmod N$  for all  $a \in \mathbf{Z}$ . Solovay and Strassen proved that the congruence fails for at least half of  $\{1, \dots, N-1\}$  when  $N$  is composite and not a prime power, however. So their algorithm involves checking  $k$  randomly chosen integers  $a$  with  $0 < a < N$ . The authors remark that, although a polynomial-time probabilistic algorithm for factoring *polynomials*, due to Berlekamp, existed before Solovay–Strassen, it was only the latter that sparked the Computer Science community’s interest in probabilistic algorithms. Thus, even in this one part of the book that is not concerned with polynomials, Von zur Gathen and Gerhard manage to promote a theme of their book: computer scientists need to take algorithms that compute with polynomials more seriously.

Chapter 19 begins with a review of the Cunningham project for factoring large integers including the so-called Fermat numbers. A typical factorization algorithm with input  $N$  presupposes that all prime factors below  $10^6$ , say, have been removed. Further, some probabilistic primality test will already have been applied so that  $N$  can be assumed composite. Generally speaking, the algorithm will return one nontrivial divisor  $d$ , and a complete factorization of  $N$  will be obtained by applying the complete procedure—including removal of small prime factors and primality testing—recursively to  $d$  and  $N/d$ . Von zur Gathen and Gerhard consider only probabilistic algorithms.

Also, all the algorithms discussed assume that input  $N$  is not a prime (or perfect) power. For most of the algorithms, a proof of correctness is provided. (The one exception is Pollard’s  $\rho$ -method, for which no such proof is currently known.)

- The first algorithm considered is the Pollard–Strassen method for finding  $N$ ’s smallest prime factor below some bound  $b$ , should such exist. A complete factorization of  $N$  is then obtainable in  $O^{\sim}(N^{\frac{1}{4}})$  word operations. This algorithm essentially involves computation with polynomials in  $\mathbf{Z}[x]$ , specifically, multipoint evaluation.
- Pollard’s  $\rho$ -method is the topic of §19.4. Floyd’s Cycle Detection Trick and a recursively defined sequence in  $\mathbf{Z}_{\mathbf{N}}$  are used to obtain the smallest prime factor of input  $N$  with an expected running time of  $O(\sqrt{p} \cdot \mathcal{M}(\log N) \cdot \log \log N)$ . Recursive application produces a complete factorization of  $N$  in  $O^{\sim}(N^{\frac{1}{4}})$  word operations, expected case.

Chapter 20, entitled “Application: Public Key Cryptography,” is a brief survey of six public-key cryptosystems including RSA and Diffie–Hellman. Since factorization of an integer  $N$  of more than a couple hundred digits is beyond the range of available big integer software, cryptosystems whose security depends upon the presumed hardness of factoring such an  $N$  are described by the authors as “applications.” But they apparently do not mean that factoring algorithms, in particular, are being applied, despite the positioning of this chapter. That is a good thing, since the encryption/decryption functions of several of the systems considered do not involve factoring anything.

## 2.5 Part V (Hilbert)

Chapter 21 describes Gröbner bases for polynomial ideals and two application areas: implicitization of algebraic varieties and solving systems of polynomial equations. Two other application areas are introduced in Chapter 24: logical proof systems and analysis of parallel processes. Chapter 22 discusses an algorithm for symbolic integration that, given an expression for rational function  $f(x)$ , returns one for  $\int f(x)dx$ . Chapter 23 gives an algorithm for symbolic summation: given an expression for  $g(n)$ , it returns a closed form expression for  $f(n) = \sum_{0 \leq k < n} g(k)$ .

## 3 Opinion

As indicated by the authors themselves, the intended audience includes both advanced undergraduates and graduate students in an algorithms course. Readers will need to be familiar with a fair amount of the theories of groups, rings, and fields, as well as linear algebra. Such mathematical preliminaries are covered quickly, but thoroughly, in an appendix (actually the final Chapter 25) of the text. This is in contrast to textbooks in which the needed material from group theory, say, is reviewed in the body of the text just prior to its first use. The latter device, however useful in a text whose development of topics is largely linear and cumulative, is less appropriate for a text such as **Modern Computer Algebra** that is unlikely to be read cover-to-cover. Having said this, it seems likely that most readers would rarely need to consult the appendix, since definitions of the concepts and structures introduced are often reviewed within the text itself.

Before introducing a new application, the authors provide abundant examples of contexts in which it might be useful. For example, in Chapter 4, before showing how computing with remainders (modular arithmetic) can be approached via EEA, the authors illustrate how it can figure in error-checking techniques for fast multiplication algorithms with large input values. Another application context that is mentioned is testing for the identity of databases at mirror sites.



The text has been assembled with extraordinary care and, for a book of this nature, has remarkably few errors—typographical or otherwise. An on-line errata list included nearly all the errors that this reviewer spotted. Notes at the end of each chapter illuminate the historical background of the ideas and algorithms presented and provide pointers into the more recent literature. Algorithms are presented in a clear ←-style pseudocode.

A numbering system whereby a lemma, theorem, and algorithm, introduced (in that order) in Chapter  $n$ , are assigned designations “Lemma  $n.1$ ,” “Theorem  $n.2$ ,” and “Algorithm  $n.3$ ,” makes for easy reference. The book’s index is excellent and incorporates a name index that points into the end-of-chapter notes. There is also an exhaustive symbol index. The clarity of this navigation system means that the book can function as a work of reference, and one expects it become one of the standard references in the field of computer algebra if this has not happened already.

There are many exercises at the end of each chapter. The ordering of these exercises follows that of the chapter, and more difficult exercises are placed last. Considerable mathematical sophistication is assumed, however. Often enough the very first exercise involves proving some number-theoretic result, say, by mathematical induction. But there is also a wealth of exercises that involve applying the algorithms discussed in the text.

The bibliography provided by Von zur Gathen and Gerhard is extensive—even scholarly, and, in general, their book is informed by a sense of the rich history of mathematics and of algorithms in particular. This reviewer knows of no text on any level that better conveys one’s sense that algorithms and the algorithm concept have made a difference for culture.

Review<sup>3</sup> of

**The Discrepancy Method —Randomness and Complexity**

Author of book: Bernard Chazelle

Cambridge University Press, 460 pages, hardcover, \$64.95

ISBN: 0521770939

Author of review: Jin-Yi Cai, University of Wisconsin, Madison

Complexity Theory is the study of quantitative limitations of computation. The central question in Complexity Theory is what can and what cannot be solved by efficient algorithms, and most importantly why certain problems are difficult to solve. For certain problems, provably efficient deterministic algorithms seem to be lacking while efficient randomized algorithms do exist. Sometimes, however, efficient randomized algorithms have been found first and then efficient deterministic algorithms are found afterwards, often by a certain process of derandomization.

One primary focus of current Complexity Theory research is: To what extent is randomization essential in computation? This is a problem the Complexity research community is likely to be occupied with for a long time to come. It is a quest that is both extremely deep technically as well as profound philosophically. The book by Bernard Chazelle, the leading computational geometer of our day, is a masterful exposition of some of the most beautiful aspects of this theory.

The central theme of the book, as the title implies, is Discrepancy Theory, and its applications to the theory of computing, especially to derandomization. From the moment I laid my hand on this book, I had no doubt in my mind that Professor Chazelle has produced a masterpiece that few books published in our field can equal. It is not as encyclopedic as the multi-volume treatise

---

<sup>3</sup>©2002 Jin-Yi Cai

by Don Knuth; but my admiration for it is such that, the volumes by Knuth are the only other scholarly books published in computer science that I can think in comparison.

From the choice of topics included, to its masterful exposition, I stand in awe of the beauty and elegance of the book, and, above all, the *exquisite taste* of the author. In about 400 pages, excluding bibliography and 3 short (and very elegant) appendices (on Probability, Harmonic Analysis and Convex Geometry, respectively), the book managed to lead us on a magnificent tour of many facets of Discrepancy Theory and applications. These topics include Pseudorandomness, Communication Complexity, rapid mixing Markov Chains, Modular forms, Geometric sampling, VC-dimension Theory, Voronoi diagram, derandomization, linear programming, linear circuit complexity, etc. From the foremost computational geometer of our day, it is also fitting that the book features a tour de force by the author himself, of an optimal deterministic algorithm for convex hulls in all dimensions. The book also presents, as far as I know for the first time in book form, another crowning achievement by the author himself, a deterministic minimum spanning tree algorithm with running time  $O(m\alpha(m, n))$ , where  $\alpha$  is the classical inverse Ackermann function. Both these algorithms are achieved by suitable (and intricate) derandomization based on Discrepancy method. In the minimum spanning tree problem, there is still potentially a gap, for a linear time randomized algorithm is known due to Karger, Klein and Tarjan, which further relies on a linear time verification procedure for MST, an idea goes back to Komlós. The convex hull problem is perhaps the most famous problem in computational geometry, and the study of the minimum spanning tree problem predates the beginning of the field of computer science itself. The various featured topics might be eclectic, but here you will see so many beautiful ideas come together, and come alive, and you can do nothing but be swept away by the sheer elegance.

Given a set system with  $n$  points and  $m$  subsets, can you color the points to two distinct colors, such that every subset contains roughly an equal number of points with each color? How close can it be? How close can it not be? How good a coloring can one get with an efficient procedure? How about points and subsets with extra regularities such as distributed in Euclidean space? These are the basic questions the Theory of Discrepancy deals with. Often one strives to achieve deterministically what can be shown to hold under uniform or carefully chosen distributions. It is therefore not surprising that the subject has a lot to do with derandomization in computation, and also, certain tools such as Fourier Analysis play an important role here.

The book starts off with a succinct introduction to combinatorial discrepancy theory. You will meet the greedy methods including the method of conditional expectations, and the hyperbolic cosine algorithm. You will meet the entropy method, the Beck-Fiala Theorem, and VC-dimension. You will also be introduced to the Hadamard matrix, eigenvalue bound, and a classical theorem of Roth (the Fields Medalist), and summed up from the point of view of Harmonic Analysis.

The second chapter deals with upper bound techniques in Discrepancy Theory. We see Halton-Hammersley Points, Ergodicity of Arithmetic Progressions on a circle, Weyl's Criterion, Quaternions and  $SO(3)$  (the special orthogonal group in dimension 3), Spherical Harmonics and the Laplacian, Hecke Operators and the Ramanujan Bound. Also included, is a nice introduction to Modular Group, Modular Forms, Zeta functions and  $L$ -functions, with a glimpse of Hasse-Weil, Shimura-Taniyama-Wiles. The author modestly insists that he is "hardly an expert on this", and offers the following: "... apology to Oscar Wilde, he is always ready to give to those who are more experienced than himself the full benefits of his inexperience." I found his little tour of this most sanctified realm of Arithmetic Algebraic Geometry beautiful and refreshing. Of course, with a mere few sections, one can not expect any full proofs. But here, without all the proofs, I found the exposition very appealing intuitively and it gives a very good sense of the big picture. (Of course here I should offer my own apology to Oscar Wilde, squared.)

In chapter 3, the author discusses lower bound techniques. We will see the Method of orthogonal functions. We see the proof of the theorem that states that the mean-square discrepancy for axis-parallel boxes is at least  $\Omega(\log n)^{d-1}$  for any set of  $n$  points in the unit square of dimension  $d$ . We see Haar Wavelets applied. The next section relaxes the restriction to non-axis-parallel boxes. We meet Beck's amplification method, Bessel functions and Fejér Kernel. The chapter finishes off with the finite differencing method.

Chapter 4 deals with Sampling, how to extract a small set of representatives from a large data set. It introduces  $\epsilon$ -Net and  $\epsilon$ -Approximation, Sampling in bounded VC-dimension. Along the way it also gives a primer on Hyperbolic Geometry. Some of the sampling techniques are best viewed through the prism of Hyperbolic Geometry. As the author indicates, while it is possible to translate these constructions back to Euclidean geometry, "this would be a mistake, . . . , (for) much of the poetry would get lost in the translation". Now, how long ago have you seen a computer science book where the "poetry" of the sublime is of such high priority? Speaking of language, this book uses the word "Method" often (and early! It is in its title.) Such profound insight and depth are codified by the author with this somewhat modest but respectable word "Method", that it just brings a smile to my face whenever I think of the great world of Object Oriented Programming, where words like "Class" and "Method" occupy such an exalted place.

Chapter 5 deals with some Geometric Searching Problems. Chapter 6 deals with complexity lower bounds, especially arithmetic circuits in range searching. Eigenvalue, eigenspace, and entropy are the main tools (the combinatorial matrix rigidity approach of Valiant is not included). There is also a section on geometric databases. Chapter 7 is on Convex Hull and Voronoi Diagrams. Here the optimal deterministic algorithm for convex hull in all dimensions by the author is presented. The presentation follows a simplified form given by a joint work of Brönnimann (a former student of Chazelle), Chazelle, and Matoušek. Chapter 8 deals with linear programming. Also you will see Löwner-John Ellipsoids. Chapter 9 is on Pseudorandomness. We see finite fields and character sums, pairwise independence as a replacement for total independence, universal hash functions, random walks on expanders, pseudorandom bits from quadratic residuosity, and finishes off with a section on polynomials, small Fourier coefficients and low-discrepancy arithmetic progressions. Chapter 10 is on communication complexity. We see the matrix rank bound, and much more. Chapter 11 is on the Minimum Spanning Tree Algorithm mentioned at the beginning. We start with linear searching as low-discrepancy sampling and soft heap, a data structure invented by the author. Then we are onto his glorious deterministic algorithm on Minimum Spanning Tree.

A book of art, a book of love, this book belongs on the shelf of every theoretical computer scientist with a discriminating taste. As I stated earlier, the only other published computer science books I can think of to compare it to are the volumes of Don Knuth. I am confident that we will all agree 25 years from now.

Review of: **Computability and Complexity Theory** <sup>4</sup>  
by Authors: Steven Homer and Alan L. Selman  
Publisher: Springer

Review of: **The Complexity Theory Companion**  
by Authors: Lane A. Hemaspaandra and Mitsunori Ogihara  
Publisher: Springer

Reviewed by E W Čenek, University of Waterloo

## 1 Overview

The study of computability and complexity theory lies at the root of computing science, as computer scientists try to address the dual questions of which problems are in fact solvable, and how much effort should we expect to expend to solve a given problem.

Problems are grouped together in classes on the basis of these questions. Thus P is the class of problems that can be solved in polynomial time, while NP is the class of problems that can be accepted in polynomial time using a non-deterministic Turing Machine. NP-complete problems were first studied in the sixties and early seventies.

## 2 Summary of Contents

### 2.1 Computability and Complexity Theory

Homer and Selman's "Computability and Complexity Theory" is clearly intended as an introductory text. The book is reasonably self-contained, and tightly focused on its topic.

The book can loosely be divided into three parts; the first and shortest comprises the Preliminaries; a brief discussion of the different concepts which will be used in the other chapters, including words and languages, representations, graphs, logic, cardinality, and elementary algebra.

The second part comprises a discussion of computability. Turing machines are introduced in Chapter 2, including discussions of multi-tape and non-deterministic Turing Machines. Chapter 3 concentrates on the issue of computability, and includes discussion of the Halting Problem (and halting problem reductions), the S-m-n theorem, and Rice's Theorem. Section 3.9 explicitly discusses Oracle Turing Machines, which are at the root of complexity theory via Church's Thesis.

The third and largest part comprises the discussion of complexity. The first three chapters are leading straight up to this material, and Homer and Selman do not disappoint. After a short chapter giving definitions of the varying complexity classes, Chapter 5 focuses on the properties the complexity classes have in common, including issues such as simultaneous simulation and reductions of the numbers of tapes used. This chapter also discusses all known inclusions between complexity classes, and discusses complements of complexity classes.

Chapter 6 focuses on the two specific classes P and NP, characterizing P and NP, and delving into the notion of NP-completeness. It includes a proof of the Cook-Levin theorem that SAT is NP-complete. Chapter 7, in contrast, concentrates on the relative complexity of problems, introducing the notion of problems which are NP-hard. Intuitively, NP-hard problems are those problems that are not necessarily in NP but are at least as difficult as every problem in NP. As well, the structure

---

<sup>4</sup>© E. W. Čenek, 2002.

of NP and NP-P is touched upon, and lastly includes a brief discussion on complete problems in the other complexity classes.

## 2.2 The Complexity Theory Companion

Hemaspaandra and Ogihara take a very different approach in “The Complexity Theory Companion”. They start from the viewpoint that simple algorithms lie at the heart of complexity theory, and develop the topics from there. Each chapter is devoted to a simple algorithmic gem (or idea), which is then used to develop several complexity results, and to motivate open issues.

This book is very definitely not an introductory text to complexity, and assumes that readers are comfortable with the different complexity classes, leaving definitions and discussion of these to appendices at the end of the book, so that I inserted a permanent bookmark there while reading.

Rather than starting with preliminaries, the authors wade right in, in Chapter 1, with the discussion of sparse sets and the (proven) gem that there are no sparse NP-complete sets unless  $P=NP$ . In turn, following chapters are devoted to these ideas and techniques:

- The One-Way Function, including both definition and ramifications of the existence of such functions. Including the ramification that one-way functions exist if and only if  $P \neq NP$ .
- The Tournament Divide and Conquer, where the ubiquitous divide and conquer approach is used to prove that P-selective sets have small circuits. A set is P-selective if there exists a polynomial time selection algorithm that, given two inputs, chooses the input that is logically more likely to be in the set. Specifically, if only one of the two inputs belongs to the set, that input will be selected.
- The Isolation Technique, which tries to use the idea that the study of NP would be far simpler if each problem in NP had exactly one unique solution<sup>5</sup>. Using randomness, it is possible to show that any problem in NP is randomly reducible to USAT (SAT with only one unique solution.)
- The Witness Reduction technique tries to reduce the number of accepting paths of non-deterministic machines. The converse, adding accepting paths, is obviously trivial. Interestingly, the effect of this technique is to show complexity class collapses.
- Polynomial Interpolation and the interactive proof system; the problem is converted into arithmetic formulas, and the formulas are verified using a gradual, random instantiation of variables. Then, for example, the class of languages with interactive proof systems is exactly PSPACE.
- The Nonsolvable Group Technique; using the nonsolvability of the permutation group  $\{1, \dots, 5\}$  to show the power of polynomially-sized bounded-width branching programs. One result is that polynomial-size 5-width branching programs = nonuniform-NC<sup>1</sup>.
- The Random Restriction Technique; the technique of randomly restricting some values is applied to circuits, and these circuit results are used to construct oracles. The approach focuses on oracles that show that some property Q does not hold as opposed to oracles that show the property Q does hold. This is used to prove several lower bounds, including an exponential-size lower bound for parity.

---

<sup>5</sup>For instance, each graph would have exactly one or no Hamiltonian Cycle.

- The Polynomial Technique; proving closure properties of PP (and related complexity classes) by constructing gap functions, a special kind of counting function which counts the difference between the number of accepting and rejecting computation paths of nondeterministic Turing machines.

### 3 Opinion

Both books discuss complexity theory, but base their approach on different goals and different underlying principles.

Homer and Selman concentrate on a fairly straightforward introduction of computation and complexity which uses the standard approach of introducing Turing machines, and then using those to discuss the complexity of problems in P and in NP. I would have liked to see a few more examples, particularly of reductions, but in general this is a book that flows easily and quickly from topic to topic.

Hemaspaandra and Ogihara take a diametrically opposed approach, choosing to concentrate on underlying techniques and algorithms rather than on introducing and dissecting specific complexity classes. It is assumed that the reader is already familiar with the various complexity classes, so that it is possible to concentrate on the various techniques. Each of the chapters is reasonably self-contained, although I found myself frequently going back to the appendix for definitions.

As textbooks, the Homer-Selman book provides a reasonable introduction to complexity theory, hitting all the highlights. There was, however, a measurable gap between the Homer-Selman book, and the Hemaspaandra-Ogihara book, which I read next. I had to spend a lot of time in the appendices of the latter book, to fill up holes. As such, the Hemaspaandra-Ogihara book is clearly intended for people who are already comfortable with the field, and are looking for a new perspective.

Review of  
**Mathematical Theory of Domains** <sup>6</sup>  
**Cambridge Tracts in Theoretical Computer Science #22**  
**Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor**  
**Cambridge University Press, 1994**

Reviewer: P. Daniel Hestand

### 1 Overview

When Dana Scott was explicating the foundations of denotational semantics, he laid the groundwork for the subject of this book: domain theory. Domain theory is an integral part of denotational semantics and not study of denotational semantics could proceed without an introduction to domains. Domains are generally presented without explanation and for most cases this suffices. Part of the reason little explanation of domains is given is that there is a scarcity of material collected into a coherent text. Some of the available works on domain theory are by now quite old (ca. 1970's) and some are just plain hard to get (the "Pisa Notes" by Gordon Plotkin). Domains are alluded to in other fields, most notably, mathematical physics [1] but again no explanation of the theory is given. This book represents an attempt at collecting the theory of domains into an updated text, delving into the inner workings of domains, and showing how they fit into mathematics

---

<sup>6</sup>© IONA Technologies, PLC, 2002

and computer science in general.

So what are domains? In short, domains are complete partial orders with a least element. What this means is that they are a set of objects with strict ordering relation. The ordering relation is usually written as  $a \sqsubseteq b$  where this is interpreted to mean that  $a$  has more informational content than  $b$ . If this sounds familiar, it should, because this is identical to the notion of a lattice and the underlying concepts of recursion theory.

## 2 Summary of Contents

The book is divided into two parts. The first part introduces domain theory via a discussion of domains and their representations, and the solution of equations involving domains and their representations. The second part ties the previously presented theory into Computer Science through illustrative uses and connections. The first seven chapters constitute part one and remaining five make up part two.

- Preliminaries — The preliminaries introduce the basic notation and concepts of set and category theory that are the foundation for everything that follows. Obligatory material, but I found myself referring back frequently.
- Fixed Points — Fixed points are introduced here to motivate the study of domains. This section begins with the example of a GCD computation as a program and asks the questions
  1. What does a program explicitly define?
  2. What does a program implicitly compute?
  3. How are the previous two questions related?

The authors then show how the GCD program can be considered to compute the least fixed point of the GCD functional and then proceed to define functionals. The chapter concludes with the introduction of  $\omega$ -complete partial orders.

- Complete Partial Orders — This chapter introduces complete partial orders as a minimal model of computation and defines the notion of continuity of cpo's. Examples of continuous cpo's are given through the Cantor space and the Baire space. The chapter concludes with the various constructions one can make using cpo's such as Cartesian products and function spaces as the more common constructions and other simpler constructions such as smash product, smash sum, and lifting.
- Domains — This chapter and the four that follow represent the heart of the book. Scott-Ershov domains or “domains” are introduced as algebraic cpo's which are consistently complete. What this means is that domains are cpo's in which each element is the limit of its compact approximation. Consistent completeness is required for Cartesian closure. The first section provides the basic definitions for domains. The next section provides the Representation Theorem for domains and introduces the concept of a conditional upper semi-lattice with least element (cusl). The theorem states that, up to isomorphism, a domain is uniquely determined by its compact elements. The chapter closes with the various constructions on domains with particular emphasis on Cartesian closure.

- Domain Equations — Given that we know what a domain is and some operations that we can perform on domains, the next question we should ask is “How can we solve equations involving domains?” This chapter is an introduction to solving domain equations. More specifically, this chapter addresses whether or not the solution to a given equation involving domains is itself a domain and if so, do fixed points exist. The first few sections of the chapter introduce concepts essential to understanding domain equations such as least fixed points,  $\omega$ -continuity, subdomains, projection pairs, and direct limits. The final result of the chapter is a theorem that states that if we have an  $\omega$ -continuous functor on a domain, then that functor has a least fixed point. The proof is non-constructive and simply illustrates that solutions exist for domain equations.
- Topology — This chapter is an interlude to introduce some basic topological concepts such as continuity, separation axioms, and compactness. These concepts are used in constructive definitions of solutions to domain equations in the remaining chapters.
- Representation Theory — The focus of this chapter is to present several different representations of domains for which solution of domain equations is possible. The authors present algebraic, logical, and topological representations. The three are precusl’s (weaker versions of cusp’s), information systems, and formal spaces. Accompanying each representation are the constructions as discussed in the chapter on domains. The chapter also discusses the relationship between the categories of the three representations. The concluding section is the important result: how to solve domain equations up to identity. This is done using the precusl representation.
- A Universal Domain — This chapter presents the notion of a universal domain or a countably-based domain in which every other countably-based domain can be embedded. This provides yet another method for solving domain equations. However, this method is only effective for equations involving countably-based domains such as are used in the study of the semantics of programming languages. The end result is an encoding of the solution to the domain equation which must be decoded to retrieve the domain.
- Representability in Domains — This chapter marks the beginning of Part II on Special Topics. The chapter is a discussion of how domains can be used to study other structures of mathematical interest. Ultrametric spaces are the first example where the authors show how ultrametric spaces may be embedded in domains. This embedding allows for the use of domain theory to study the ultrametric spaces. The authors then move to a discussion of the total elements of a domain and continuous functionals of finite type as introduced by Kleene and Kreisel.
- Basic Recursion Theory — This chapter presents recursion theory in the context of  $\mu$ -recursive functions and presents the basic results for these. The chapter is a prelude to the next chapter on effective domains. However, the final sections of the chapter present an enlightening argument to show how the partial  $\mu$ -recursive functions can be obtained by from very simple functions using substitution and the fixed point theorem for computable functionals.
- Effective Domains — This chapter introduces the notion of effective or computable domains. The first sections of the chapter discuss effectiveness in general and for cusp’s. The third section then uses this to define and introduce results for effective domains. The fourth section discusses constructive subdomains which are subdomains containing only computable



elements. The last sections are generalized restatements of two key theorems, namely the Myhill-Shepherdson Theorem and the Kreisel-Lacombe-Shoenfield Theorem.

- **Power Domains** — Power domains are introduced in this chapter to provide context for studying non-deterministic or parallel programs. Power domains were originally introduced by Plotkin in 1976 to give a semantics for these types of programs. The main gist of the chapter is to show a class of objects called SFP-objects (for sequence of finite partial orders) that are the largest subcategory of countably based algebraic cpo's that is cartesian closed. The chapter closes with an algebraic characterization of SFP-objects.
- **Domains as Models of Formal Theories** — This final chapter is a discussion of domain theory as it applies to the lambda calculus and to denotational semantics. The latter concept is the context in which Scott introduced domain theory and so this chapter represents a return to the original motivation for studying domain theory in the first place. The chapter opens with an introduction to the  $\lambda$ -calculus and quickly moves into models of the  $\lambda$ -calculus. As this section notes, the principle difficulty with modeling a theory like the  $\lambda$ -calculus is that one is able to apply any term to itself. This makes it difficult due to the basic limitation in finding a structure in which one can interpret closed terms as both an object of the structure and as an operation on objects in that structure. This motivates a discussion of applicative structures. From these discussions the authors then move to domain interpretations of the  $\lambda$ -calculus and introduce a simply typed  $\lambda$ -calculus. The final section is a brief discussion on parametrizations which are used to provide a framework for interpreting type expressions in e.g., second-order typed  $\lambda$ -calculus and Martin-Löf's intuitionistic theory of types.

### 3 Opinion

This book was difficult to read in a single pass - in fact, I read it twice. There is considerable material that must be read between the lines to be able to truly understand the theory of domains. I found that I used the references a considerable amount to provide the context for some of the theorems and proof suggestions in the text. However, the material in this book was extremely useful for providing a basic understanding of the theory underlying denotational semantics and gave me a fresh view of denotational semantics. The exercises in this book were of sufficient difficulty that this book would make a useful textbook for a graduate class in domain theory and I can imagine that one could cover the entire book in a single semester. The exercises frequently introduced additional concepts that extended the material in the text but were not necessary for an understanding of the basic theory. If you have an interest in the foundational material for denotational semantics and an alternative interpretation of some of the basics of theoretical computer science then I heartily recommend this book to both read, keep and use.

### References

- [1] Geroch, Robert. **Mathematical Physics**. University of Chicago Press, Chicago, 1982.