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`

Welcome to the Book Reviews Column. We hope to bring you at least two reviews of books every month. In this column four books are reviewed.

1. **Gems of Theoretical Computer Science** by Uwe Schönig and Randall Pruim. Reviewed by: Danny Krizanc. This book is a collection of articles that showcase theorem(s) that the authors think are particularly interesting.
2. **Network Design: Connectivity and Facilities Location, DIMACS Workshop** edited by Pardalos and Du. Reviewed by Boris Goldengorin. This is a collection of articles on Network Design that were presented at a DIMACS workshop. The intent is to give an up-to-date view of the field.
3. **The Optimal Implementation of Functional Programming Languages** by Andrea Asperti and Stefano Guerrini. Reviewed by Christopher League. This is about issues that arise (and how to resolve them) when trying to implement a functional programming language.
4. **Indiscrete Thoughts** by Gina-Carlo Rota. Reviewed by William Gasarch. This is a collection of essays on math, mathematicians, and related topics. Some of his insights area applicable to Computer Science as well. This review originally appeared in *Journal of Logic and Computation* Vol. 9, No. 4, Aug 1999, pages 595-596. It is reprinted here with permission.

[1]© William Gasarch, 2000.

Review of

Gems of Theoretical Computer Science[2]

Authors: Uwe Schöning and Randall Pruim

Publisher: Springer-Verlag, 1998

ISBN 3-540-64425-3

Hardcover

Price: $39.95 (US)

Reviewed by: Danny Krizanc
Wesleyan University, Middletown, Connecticut.

1 Overview

In the summer of 1993, Uwe Schöning taught a course at the Universität Ulm on concepts in theoretical computer science. Rather than taking the standard broad survey approach, Schöning's method was to choose a number of important theorems ("outstanding results") and present them in depth with complete proofs including false leads and important implications. His goal was to have "the students understand and sense 'how theoretical research is done.' " The resulting course materials were later put together and made into a book in German: "Perlen der Theoretischen Informatik." Randall Pruim has "translated, revised and expanded" that text to give us "Gems of Theoretical Computer Science."[3]

After some introductory material that briefly reviews the most important definitions and results required to understand what follows, the rest of the book consists of 26 chapters, each devoted to a single "gem." The format of each chapter is optimized for achieving Schöning's goal of showing "how theoretical research is done." A chapter usually begins by giving a short informal description of the topic to be considered, how it fits in to theoretical computer science along with motivation for its study. This is followed by any definitions required to formally state the results explored in the chapter. Next comes an exposition of the main theorem and its proof. Unlike standard presentations, the format here includes many exercises that must be solved before continuing. (Solutions to all exercises are provided at the back of the book.) The reader is encouraged to attempt all exercises by interspersing easy with difficult ones and giving insightful hints for the most difficult ones (sometimes all but giving away the answer). Also distinct from standard texts, asides are often followed in depth and a number of consequences of the results are completely worked through. At the end of each chapter we find an annotated set of references for those interested in history, other presentations of the same material and related results.

2 Summary of Contents

The gems Schöning and Pruim have chosen for us are mainly from logic, computability, complexity theory and circuit theory. The topics of the 26 chapters are

[2]© Danny Krizanc, 2000

[3]Not having access to the original and not knowing German in any case, I will not comment on the translation or revisions other than to say that the book does have the "feel" that Schöning suggests he was going for in his preface to the original. As for the expansion, Randall Pruim has added two "gems" of his own (chapters 25 and 26) and has done an admirable job of capturing the same "pleasure of the pursuit of understanding" present in the earlier chapters.

1. The priority method and its use in solving “Post’s Problem” of whether there exist undecidable, computably enumerable sets that are not Turing equivalent to the halting problem.

2. Hilbert’s Tenth Problem: whether there is an algorithmic method for solving Diophantine equations.

3. The equivalence problem for LOOP(1) and LOOP(2) programs.

4. The second LBA problem: whether the class of languages accepted by nondeterministic linear space-bounded Turing machines (LBAs, linear bounded automata) is closed under complement or not.

5. Random walks on graphs, universal traversal sequences and their relation to the study of logarithmic space-bounded Turing machines.

6. An exponential lower bound for the length of resolution proofs of the pigeonhole principle.

7. Characterizing the set of all sizes of finite models of a given formula (the spectral problem) and its relation to the P versus NP question.

8. Applications of Kolmogorov complexity to the universal distribution and average-case complexity.

9. Applications of Kolmogorov complexity to lower bounds arguments and the Prime Number Theorem.

10. Probabilistically approximately correct learning and Occam’s razor.

11. Lower bounds on the circuit complexity of the parity function using the “random restrictions” method of Furst, Saxe and Sipser.

12. Lower bounds for parity using the algebraic techniques introduced by Razborov and Smolensky.

13. The Craig Interpolation Theorem (for any two formulas F and G in propositional logic such that $F \to G$ there is a formula H which uses only variables occurring in both formulas such that $F \to H$ and $H \to G$) and its relation to the P versus NP question.

14. Testing the equivalence problem of one-time-only branching programs is in *co*-RP.

15. The Berman-Hartmanis Conjecture (all NP-complete languages are pairwise polynomial-time isomorphic to each other) and its relation to sparse NP sets (i.e., Mahaney’s theorem).

16. Polynomial-size circuits for NP implies the collapse of the polynomial-time hierarchy.

17. Amplifying probability, recycling random bits and relations between probabilistic complexity classes and the polynomial-time hierarchy.

18. The complement of Graph Isomorphism is in $BP \cdot NP$ (bounded error probability NP) and its implications.

19. Toda's theorem: the polynomial-time hierarchy is contained in $BP \cdot \oplus P$ (bounded error probability Parity P).

20. Introduction to interactive proof systems and the class IP, including a zero-knowledge protocol for Graph Isomorphism.

21. $IP = PSPACE$.

22. $P^A \neq NP^A$ with probability one for a randomly chosen oracle A.

23. The construction of linear size superconcentrators.

24. Pebble games and the relationship between deterministic time and space.

25. Levin's theory of average-case complexity.

26. Grover's quantum algorithm for searching a database.

3 Opinion

At times the book shows some of the weaknesses of having its source in course materials (e.g., some repetitions, inconsistent notation, typographical errors) but these minor inconveniences are easily overlooked. They are far outweighed by the strongest feature of the book: its format. It is clear that a lot of work has gone into choosing just the right exercises to expose the inner workings of each proof. This makes it ideal for self-study by graduate students (or talented undergraduates) new to the material or experienced researchers looking to expand their understanding of something they were aware of but had never studied in depth. As somone who works mainly in algorithms, most of these topics were either new to me or I hadn't considered them since leaving graduate school. As such, I found the book to be a perfect oppurtunity to pick up topics I had missed or to refresh my memory of things long forgotten. While it is always possible to quibble with the choice of topics (everyone will have at least one or two favorite results left out and one person's gem is often a piece of coal to another), the book would make a good text for an advanced graduate course on computability and complexity for students interested in pursuing research in these areas. An interesting project for such a course might be to have the students develop their own chapters from a gem of their choosing. With any luck this book will inspire imitations in other areas of computer science.

Review of
Network Design: Connectivity and Facilities Location
(Proceedings from DIMACS workshop in April 1997
Publisher: AMS 1998
ISBN 0821808346
$79.50, Hardcover

Reviewed by Boris Goldengorin University of Groningen, Dept of Econometrics and Operations Research, The Netherlands.

Positive impressions:

1. *The title of this book and included papers*: all papers really correspond to the title of this book except the paper "A case Study of De-randomization Methods for Combinatorial Optimization" by José D.P. Rolin and Luca Trevisan. Anyway, this paper reflects one of the general topics in "Discrete Mathematics" and devoted to the analysis of the *conditional probabilities method* for routing, packing and covering integer linear programming problems. It is well known that network design problems can be formulated as an integer programming problem, the relaxation over the reals of which is usually used to develop approximations algorithms. For an integer linear programming formulation we usually construct a pair of feasible solutions (primal and dual relaxation) and take the ratio between the values of these primal and dual solutions as a constant r for the corresponding r-approximation algorithm. For an integer mathematical programming formulation we use a pair of the relaxed and rounded solution. If the ratio of the values of these rounded and relaxed solutions is bounded by a constant r then it tells us that the rounded solution is r-approximate. In the second case rounding a fractional solution is a difficult task. Authors have used *randomized rounding method* and have proved their approximation guarantee by using Chernoff bounds. By using the so called *pessimistic estimators* authors de-randomize their algorithm. Both of above mentioned methods can be considered as authors' contribution in theoretical computer science.

2. In papers by Siu-Wing Cheg; Dietmar Cieslik; Cees Duin; Andreas Eisenblätter; Thomas Erlebach, Klaus Jansen, Christos Kaklamanis, and Pino Persiano; Krisina Holmqvist, Athanasios Migdalas, and Panos M. Pardalos; Klaus Jansen; Sudipto Guha and Samir Khuller; Wu-Ji and J. Macgregor Smith the potential applications in Computer Science are discussed.

3. Very interesting result is announced in the paper by Marek Karpinsky and Alexander Zelikovsky. They show that the so called *ε-dense set cover problem* can be approximated with the performance ratio $c \log k$ for any $c > 0$ though it is unlikely to be NP-hard. This is in contrast to the hardness results for set cover proven by Lund and Yannakakis (1994) and Feige (1996) that the set cover problem cannot be approximated to within factor $(1 - o(1)) \ln k$ unless $NP \subseteq DTIME[k^{\log \log k}]$. Here k is the number of elements of the ground set which should be covered by its proper subsets.

4. In papers by M. Brazil, D.A. Thomas, and J.F. Weng; Siu-Wing Cheng; Dietmar Cieslik; Cees Duin; Marek Karpinsky and Alexander Zelikovsky; Narsingh Deo and Nishit Kumara; Sven O. Krumke, Madhav V. Marathe, Hartmut Noltemeier, R. Ravi, and S. S. Ravi; Madhav V. Marate, R. Ravi, and R. Sundaram; Stefan Voß and Kai Gutenschwager; David Warme; J. F. Weng; M. Brazil, J. H. Rubinshtein, D.A. Thomas, and J.F. Weng, and N. C. Wormald different generalizations of NP-hard variants for Steiner trees defined in metric and non-metric spaces are studied from

computational point of view. I think these papers should be interesting for Computer Science's people.

Negative impressions:

1. Some of authors do not indicate the relationship between Theoretical Computer Science and their contribution to this area. Usually they just point out that the corresponding problem is NP-hard (see papers by R. Battiti and A. Bertossi; M. Brazil, D.A. Thomas, and J.F. Weng). But maybe this remark is more related to Editors.

2. The paper by C. S. Adjiman, C. A. Schweiger, and C. A. Floudas is not relevant to this volume and can be considered as a paper related to some general problems in mixed-integer nonlinear programming. I think the example about optimum configuration for the heat exchanger network problem is also not very close to Discrete Mathematics and Theoretical Computer Science.

Review of
The Optimal Implementation of Functional Programming
Cambridge Tracts in Theoretical Computer Science
Published by Cambridge Press
Year: 1998, Number of pages: 392
Author of Book: Andrea Asperti and Stefano Guerrini
Author of Review: Christopher League, Yale University CS

1 Overview

Functional programming languages (such as ML, Haskell, and Scheme) treat *functions* as first-class objects. Functions can be passed as arguments, stored in data structures, and returned from other functions. Such languages encourage a programming style which emphasizes recursion and higher-order functions, rather than iteration and assignment of variables. (Side effects are banned completely in Haskell, a *pure* functional language.)

The semantics (and, often, the implementation) of functional languages are based largely on the λ-calculus, a simple rewrite system first described by Alonzo Church in 1932. Terms in the λ-calculus may be defined inductively as follows. Any variable x is a term. If M is a term, then $\lambda x.M$ is also a term, called an *abstraction*. An abstraction is analogous to a *function* in a programming language; x is the *formal parameter* and M is the *body*. Finally, if M and N are terms, then $M\,N$ is also a term. This term is called an *application*, and roughly corresponds to a *function call* in a programming language.

A term is *reduced* by repeatedly replacing sub-terms (called β-redexes) of the form $(\lambda x.M)\,N$ with $M[x := N]$, a *substitution* in which we replace all free occurrences of the variable x in M with the term N. Clearly, this operation could duplicate work: any reductions required to simplify the argument N will have to be repeated on each new copy.

Implementations of functional languages use various techniques to reduce the amount of redundant work. So-called *strict* languages (such as Scheme and ML) fully evaluate the argument N before doing the substitution. This reduction strategy, however, is non-normalizing; it is not guaranteed to reach a normal form if one exists. In contrast, *lazy* languages (such as Haskell) use a simple graph-rewriting technique to ensure that all occurrences of x in M are shared internally. Then, any reductions performed on x will update all occurrences simultaneously.

Although such sharing is used in all successful implementations of lazy functional languages, it is not *formally* optimal; duplication can still occur. Consider applying $\lambda x.((x\,y)\,(x\,z))$ to $\lambda w.((\lambda q.q)\,w)$. The argument will be shared by the two occurrences of x. When we apply x to y, however, only the first instance of $\lambda w.((\lambda q.q)\,w)$ should be involved. The standard algorithm will then proceed to duplicate this sub-term, which also (unnecessarily) duplicates the internal redex $(\lambda q.q)\,w$.

This book is about the theory and practice (but mostly the theory) of *optimal reduction*, a sophisticated graph-rewriting technique for reducing λ-terms. Here is a brief overview of the content:

- Chapters 1 through 3 provide intuition, sketch several non-optimal attempts, demonstrate the reduction algorithm (due to Lamping), and prove its correctness.
- Chapters 4 through 7 deal with connections between optimal reduction and linear logic, formal definitions of optimality and sharing, and making syntactic sense of intermediate

graphs (read-back).

- Chapters 8 through 12 turn to more practical concerns: different ways of translating λ-terms into sharing graphs, avoiding or reducing the accumulation of bookkeeping information, complexity (optimal reduction is *not* elementary recursive), language features outside the λ-calculus, and implementation issues.

I will now sketch the topics of each chapter in more depth, and conclude with my own opinion and impressions.

2 Content summary

Chapter 1 (Introduction) is really a preface; it describes the intended audience and chapter dependencies. It also gives a brief history of work in this field.

Chapter 2 (Optimal Reduction) is a friendly introduction to the problem, showing the various ways duplication can occur, and why it is so tricky to prevent. The authors survey some past non-optimal attempts at eliminating redundant computations; these include combinators, environment machines, and the sharing technique mentioned above.

Chapter 2 ends with an intuitive introduction to *sharing graphs*, and reductions on them. A sharing graph is similar to the abstract syntax tree of a λ-term, except that (1) variables are explicitly connected to the λs that bind them, and (2) there is an explicit *fan* node to express sharing. (In order for a variable to appear n times in a term, it must contain $n-1$ fans.)

The fundamental graph rewriting rules (for β-reduction and various fan interactions) are also sketched in Chapter 2. Fans interact with λ and application nodes by duplicating them. Fans interact with other fans either by annihilating (if they somehow 'correspond') or by duplicating one another. This correspondence is left completely undefined in Chapter 2.

Chapter 3 gives the full algorithm. To determine the proper pairing of fans, we must add a notion of *level* to the nodes in the graph. Fans on the same level annihilate one another. Tracking levels through reductions requires adding two new operators, a *bracket* for incrementing the levels of operators it encounters, and a *croissant*, for decrementing them. Paired brackets and croissants can also annihilate one another. Once all the operators and interaction rules have been given, the algorithm is proved correct in section 3.5 (best skipped on a first reading).

Chapter 4 explores in detail the relationship between optimal reductions and Girard's linear logic, a 'resource-conscious' logic in which assumptions are not duplicated unless explicitly deemed 'sharable'. The level-adjusting operators (bracket and croissant) in sharing graphs correspond to modality-changing operators (ϵ and δ) in linear logic. The reader probably needs previous experience with linear logic for this chapter to be meaningful.

Redex Families were proposed by Lévy in 1978 (well before Lamping developed an optimal reduction algorithm in 1989) in order to formalize the intuitive notion of shareable redexes. Essentially, redexes should be shared if they were 'created in the same way.' Chapter 5 explores several approaches to defining families: the *zig-zag* relation, causal histories, and the labeled λ-calculus. Here, we finally learn what precisely is meant by optimality.

The notion of *paths* in Chapter 6 is closely related to the concepts of the previous chapter. Achieving optimality is tricky; it is not enough to avoid duplicating reducible sub-terms. As demonstrated above, we also must avoid duplicating applications which will later *become* redexes, once some free variables are instantiated. These so-called *virtual redexes* can be formalized by defining *paths* in the graph. A path represents a virtual redex, in the sense that the path will contract (through a series of reductions) into a single edge representing the *actual* redex.

The initial encoding of a λ-term as a sharing graph looks essentially like a decorated abstract syntax tree. During optimal reduction, however, intermediate graphs generally have very little correspondence with intermediate terms in the syntactic reduction. Chapter 7 (Read-back) demonstrates a technique for computing the abstract syntax tree corresponding to an intermediate graph. This technique enables a more syntactic approach, including comparisons to well-established rewrite systems. The technique works, however, by adding so-called *propagation rules* to the system, which do not preserve optimality.

Chapter 8 marks the beginning of the more practical segment of the book. It compares and contrasts several different ways of encoding λ-terms as sharing graphs.

The next chapter (Safe Nodes) addresses the problem of accumulation of control operators (brackets paired with croissants, and fans with garbage), a primary source of inefficiency in optimal reduction. This chapter explains that *garbage collection* is a crucial component of optimal reduction. *Safe nodes* are points in the graph where operators can be eliminated safely without affecting the computation. The algorithm given for recognizing these safe nodes works well in practice, but is not itself optimal.

Chapter 10 discusses the complexity of optimal reduction. In short, "a sequence of n parallel β-reduction steps [...] cannot be bounded in general by any Kalmar-elementary function $\mathbf{K}_l(n)$[§]." This result is due to Asperti and Mairson (POPL'98); the complete analysis is not included in the book. Rather, this chapter sketches the key insights, including *superposition* (representing different values, such as true and false, in a single graph) and *higher-order sharing* (which leads to a combinatorial explosion in the size of graphs). Using these techniques, we can build worst-case graphs that support non-elementary computation in a constant number of parallel reductions.

So far, sharing graphs have been used to represent pure λ-terms only. Chapter 11 extends the technique to a richer functional programming language, including primitive data types, control structures, and recursion. To do this, the authors make a connection with Lafont's Interaction Nets.

The final chapter is devoted to the Bologna Optimal Higher-order Machine (BOHM), a prototype implementation of a core functional language using optimal reduction. It describes the source language, all the reductions and how they are implemented, and some other implementation details, such as garbage collection. Graph sizes are given (with and without garbage collection), but run times for 'typical' programs are omitted. The authors claim, however, that BOHM (which is just a prototype) is about an order of magnitude slower than call-by-value implementations (SML or Caml) and only slightly slower than lazy implementations (Haskell). Considering the worst-case complexity of optimal reduction, this is not an unfavorable result! But the key question remains: is optimal reduction worthwhile in a real implementation?

The authors point out that existing functional programs typically do not make extensive use of higher-order functionals, presumably because they would be grossly inefficient in existing implementations. Thus, these programs do not greatly benefit from the higher-order sharing achieved by using optimal reductions. (Pure λ-calculus terms, in which all data are represented as functions, benefit much more from optimal reduction.)

[§]a fixed stack of l 2s with an n on top: $2^{2^{\cdot^{\cdot^{2^n}}}}$

3 Opinion

The book is written in an accessible style, with many examples and diagrams. The authors' notation for λ-terms is somewhat non-standard: the term $\lambda x.M\,N$ should be parsed as $((\lambda x.M)\,N)$ in this book, not $(\lambda x.(M\,N))$. This can be quite confusing at first because (1) the notation is never explained, and (2) there are several typographical errors and wrongly-parenthesized terms.

Notation aside, anyone studying λ-calculus or functional programming would benefit from the friendly introduction in Chapters 2 and 3. Those interested in linear logic will find plenty of connections with optimal reduction in Chapters 4 and 6. The remaining topics, though presented well, are somewhat scattered. The authors even warn in the preface that you probably do *not* want to read the book sequentially! They suggest several different 'access paths', depending on the reader's "interests and cultural formation."

Review of
Indiscrete Thoughts
Author: Gina-Carlo Rota
Publisher: Birkhäuser, 1996
ISBN: 3-7643-3866-0
$36.50, Hardcover, 271 pages

Reviewer: William Gasarch

Indiscrete Thoughts is not a math book. It is a collection of essays *about* math, mathematicians, philosophy and philosophers. Dr. Rota has been involved with all four of these topics for quite some time. His points of view are always interesting, sometimes controversial, and have the ring of truth about them.

Part I is entitled *Persons and Places* and contains chapters 1–6. Chapters 1 and 2 are about Rota's days as an undergraduate at Yale and a graduate at Princeton. These chapters are not autobiographical– they are about the people he interacted with there. We get to read about Church, Feller, Artin, Lefshetz, Schwartz and others. The portrayals are neither negative or positive, just interesting. For example, of Alonzo Church he writes "He never made casual remarks: they did not belong in the baggage of formal logic. For example, he would not say "it is raining." . . . He would say instead "I must postpone my departure for Nassau Street, inasmuch as it is raining, a fact which I can verify by looking out the window." He writes of Stan Ulman's going from a hardworking and creative mathematician to a lazy but (perhaps more) creative one because he had encephalitis (which required brain surgery). About his post-operation condition Rota writes "Stan Ulam was lazy, but he knew how to turn laziness into elegance and conciseness of thought."

Chapter 3 has the intimidating title *Combinatorics, Representation Theory, and Invariant Theory* but it is actually about how mathematics works. His main points are

1. Mathematics is NOT about formal proofs. The formalization occurs after we know the intuition has been established and we know what needs to be true.

2. Some mathematicians are problem solvers and some are theorizers. Theorizers are more likely to be underated. He tells the story of Grassman and Exterior Algebra to illustrate his point.

Part 2 is entitled *Philosophy: A minority view* and contains chapters 7–17. Chapter 7 is entitled *The pernicious influence of mathematics upon philosophy* and was controversial in analytic philosophy circles (and made Rota unwelcome in such circles). His main point is that Philosophy has recently been trying to imitate mathematics by insisting on precise statements and clear definitions. This approach also means disgarding most of traditional philosophy (usually relegating it to Psychology, where it does not belong). In Rota's view the philosophers are wrong about both what mathematics is (they think it is about rigor and proof) and what philosophy should be (they think its goal should be precision).

Several of the essays are on Phenomenology, a philosophical movement started by Husserl in 1913. Phenomenology is "dedicated to describing the structure of experiences as they present themselves to consciousness, without recourse to theory, deduction, or assumptions from other disciplines such as the natural sciences." That quote is *not* from Rota's book, it is from a Phenomenology website at http://www/connect.net/ron/phenom.html. Rota's book is good at explaining what are the problems Phenomenology is trying to tackle, but not quite so good at telling us what phenomenology is and how it attempts to solve these problems. The problems Rota describes are as follows:

1. Once you have a proof of theorem T it seems as though Theorem T is inherent (obvious?) from the axioms. Yet Theorem T certainly does not look obvious.
2. Is mathematics created or discovered?
3. What makes a piece of mathematics beautiful?

Chapters 9 (10,11) is entitled "The phenomenology of Mathematical Truth (Beauty, Proof)." The first two of these chapters raises these questions and the third attempts to tackle them. The attempt is not altogether successful, but of course these are hard questions. The reader will find the questions and the attempt to solve them interesting.

Part III is entitled *Readings and Commentary* and contains chapters 18–21. Chapters 18 and 19 are titled *Ten lessons I wish I'd been taught* and *Ten lessons for the survival of a Mathematics department.* Both offer much good advice. I give two examples, one from each: "You will best be remembered for your expository work" and "Never compare fields" that is, never say that your field of math is better than some other field. Chapter 20 is called *A mathematicians gossip.* It is *not* scandalous material. Instead it contains short truths about how mathematics really works. I'll quote one: "Mathematicians have to attend (secretly) physics meetings in order to find out what is going on in their fields. Physicists have the P.R., savoir-faire, and chutzpah to write readable, or at least legible accounts of subjects that are not yet obsolete, something few mathematicians would dare to do, fearing expulsion from the AMS." Chapter 21 gives several "book reviews." Some are actually book reviews and others are comments on books in general (e.g., Schaums outlines are underated by mathematicians). All of the reviews have points of interest to make.

I found this book a delight to read. It should be read by every mathematician and philosopher so he can learn both what their fields are what they are not. Others such as computer scientists and physicists will also benefit from reading this fine book.