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. **Algorithmic Cryptanalysis**. By Antoine Joux. Review by Alexandre Anzala-Yamajako. This book presents algorithms that can be used to try to break break actual cryptographic constructions.

2. **Algorithmic Bioprocesses**. Edited by Condon, Harel, Kok, Salomaa, Winfree. Review by Aaron Sterling. This book is a Festschrift that grew out of a workshop celebrating the 65^{th} birthday of Grzegorz Rozenberg. The editors describe their reasons for choosing the title *Algorithmic Bioprocesses*: "the workshop did not cover the whole spectrum of natural computing research, but rather it was mostly focused on the interactions between computer science on the one hand and biology, chemistry, and DNA-oriented nanoscience on the other.

3. **Vehicular Networks, from Theory to Practice**. Edited by Stephan Olariu and Michele C. Weigle. Review by Yu Wang. Vehicular ad hoc network (VANET) is an emerging new technology to integrate the capabilities of new generation wireless networks with smart vehicles. This book explains some of the technological and social issues that arise from trying to do this.

4. **Graph Theory and Interconnection Networks**. By Lih-Hsing Hsu and Cheng-Kuan Lin. Review by Francesco Silvestri. This book covers some classical notions and results in graph theory, such as graph coloring, matching and connectivity and then, using this background, explores connectivity, fault tolerance, Hamiltonian cycles, and diagnosability. Then, these properties are discussed for some graphs, like hypercubes, star and pancake graphs, which are common in interconnection networks.

5. **Transitions and Trees: An Introduction to Structural Operational Semantics**. By Hans Hüttel. Review by Stephan Falke. In order to reason about the behavior of computer programs it is mandatory to have a precise definition of the semantics (i.e., the meaning) of the various constructs that can be used in programming languages. A formal semantics of a programming language can serve as a standard for implementing compilers or interpreters and is necessary in order verify correctness of programs. This book studies *Structural operational semantics*, developed by Gordon Plotkin in the early 1980's. It is based on transition systems that describe the evaluation steps of a program.

6. **Origins and Foundations of Computing**. By Friedrich L. Bauer. Review by Haim Kilov. This is a book on the history of computer science, that is chiefly concerned with the field before 1960. It stresses that this history goes back further than is commonly thought.

7. **Introduction to Scheduling**. Edited by Yves Robert and Frederic Vivien. Reviewed by Ben Fulton. There are many different scheduling problems depending on the parameters (e.g., number of servers, number of customers, quantity to optimize, etc). In this book, authors Yves Robert and Frederic Vivien put together a series of essays on scheduling problems, solutions, and performance guarantees, from many of the top experts in the field.

8. **Semantic Techniques in Quantum Computation**. Edited by Simon Gay and Ian Mackie. Review by Kyriakos N. Sgarbas. Even though we do not have quantum computers (yet?) researchers are building logic systems, semantic structures, even programming languages for them. This is an edited volume containing 11 lengthy articles in the broad area of semantics for quantum computation. The articles are quite diverse, some are more technical, some are more theoretical, some are more review-oriented, and some are definitely research papers.

9. **Modern Computer Arithmetic**. By Richard Brent and Paul Zimmermann. Review by Song Yan. This book is about algorithms for performing arithmetic, and their implementation on modern computers. More specifically, it collects, describes and analyzes state-of-the-art algorithms for arbitrary precision arithmetic (integers, integers modulo n, and floating-point numbers).

10. **Design of Approximation Algorithms**. By David P. Williamson and David B. Shmoys. Review by Deeparnab Chakrabarty. One way to deal with NP-completeness is to devise poly time approximation algorithms. In some cases we can even get performance guarantees such as *this algorithm will obtain a result within twice optimal*. This book is on how to design such algorithms.

BOOKS I NEED REVIEWED FOR SIGACT NEWS COLUMN
Algorithms and Combinatorics

1. *Integrated Methods for Optimization (second edition)* by John Hooke

Complexity theory and Logic

1. *Computability and Complexity Theory (2nd Edition)* by Homer and Selman.
2. *Proof Analysis: A Contribution to Hilbert's Last Problem* by Negri and Von Plato.
3. *The Universal Computer: The Road from Leibniz to Turing (Turing Centenary Edition)* by Martin Davis.
4. *Programming with Higher Order Logic* by Miller and Nadathur.

Misc Computer Science

1. *Algebraic Shift Register Sequences* by Goresky and Klapper.
2. *Information Theory and Best Practices in the IT industry* by Sanjay Mohapatra.
3. *Software Abstractions: Logic, Language, and Analysis* by Daniel Jackson.

Misc Mathematics

1. *In pursuit of the Unknown: 17 Equations that changed the world* by Ian Stewart.
2. *A Wealth of Numbers: An anthology of 500 years of popular math writings* Edited by Benjamin Wardhaugh.
3. *Applications of Combinatorial Matrix Theory to Laplacian matrices of Graphs* by Jason Molitierno.

Review of[2] of
Algorithmic Cryptanalysis
by Antoine Joux
CRC Press, 2009
501 pages, HARDCOVER

Review by
Alexandre Anzala-Yamajako anzalaya@gmail.com

1 Introduction

We easily forget how much cryptography is involved in our everyday lives. Whether we are grocery shopping with our credit card or surfing the web over a secure channel, cryptographic mechanism are put in place even if we're not fully aware of it.

A large part of getting a better understanding of a cryptosystem is trying to break it and the science of "learning by cracking" is called cryptanalysis. Cryptanalysis is in fact the main focus of this book as it attempts to present the reader with tools as well as how to apply those tools to exploit weaknesses in a cryptographic scheme.

2 Summary

The book is divided into three parts. The first part gives the reader some background on cryptography, the second part contains nine chapters each dedicated to a specific type of algorithm and the third part uses the results of the second one to show how to break actual cryptographic constructions.

The goal here is to give you an outlook as to what to expect if you open this book while not describing the content of every chapter : first it could get a bit tedious and second I'm confident that if you were interested by this summary you will love the rest.

2.1 Background

2.1.1 A bird's-eyes view of modern cryptography

This section introduces the various cryptographic notions that will be used throughout the book such as symmetric ciphers, asymmetric ciphers, MAC algorithms, signature algorithms and hash functions. The author takes great care of defining and explaining what it means for those primitives to be secure.

2.1.2 Elementary number theory and algebra background

On top of the basic math knowledge that one must have to understand the content of this book, the author also provides some interesting information such as an algorithm to solve univariate

polynomial equations or a focus on the particularity of the finite field $\mathbb{F}_{2^n}$ and its link to LFSR theory.

2.2 Algorithms

2.2.1 Linear algebra

This part introduces efficient algorithms to achieves goals such as matrix multiplication, mostly via the best algorithm we have in a practical setting : Strassen's algorithm. Also, in order to tackle issues like finding the determinant of a matrix or finding its multiplicative inverse, the author carefully describes Gaussian elimination and the various pitfalls that an implementer has to be wary of. Finally since sparse matrices arise so often in cryptography a section is devoted to linear algebra system solvers adapted to the sparse context.

2.2.2 Sieve algorithm

Eratosthenes's sieve is probably the oldest prime finding algorithm. While somewhat inefficient in its basic version the author presents several improvements to either time or memory requirements to achieve either a sublinear complexity or a small memory footprint. A presentation of the Atkin & Bernstein sieving algorithm is also given since it achieves both a sublinear running time and a memory footprint of the order of the root of the bound. Finally the author presents algorithms that allow sieving for smooth composite numbers since they are of great importance in various cryptographic context.

2.2.3 Brute force cryptanalysis

At first, it is surprising to see a whole chapter dedicated to the most naïve of approaches. However the author chose to present the subtleties of brute-force cryptanalysis and the various implementation improvements that can be made using the example of the cryptanalytic effort on the DES symmetric encryption scheme. To further legitimize the importance of brute-force optimization the author presents how it can be used as the final step of a larger cryptanalytic body of work through the example of differential cryptanalysis on the hash function family SHA.

2.2.4 The birthday paradox

The birthday can be presented as follows : we have n disjoint classes of objects, assuming objects are equiprobably distributed among classes how many objects do we need to pick until the probability of a collision is greater than $\frac{1}{2}$? The answer is of the order of $\sqrt{n}$. This fact has countless applications in the field of cryptography (the first of which being collision search for hash functions) and the author dedicates three parts of his book to treat it with details. In those three parts, one will find out about

- Generalizations of this paradox for multicollision or collisions between sets
- Efficient sorting algorithms to find the collision once we've computed a list of $\sqrt{n}$ objects
- Consequences of generating the list via the recurrence relation $X_{i+1} = f(X_i)$ where f is a random looking function and what it means for a function to look random.

- Applications of that paradox to solving the discrete logarithm problem through the baby-step giant step algorithm.

2.2.5 Fourier and Hadamard-Walsh transforms

The author introduces the Fourier transform for purposes such as fast integer or polynomial multiplication algorithms since most cryptographic implementations need efficient such primitives. Applications of the Walsh transform are found in the design of non-linear parts of secret key algorithms, the S-boxes. Without going into too much detail, the Walsh transform can help understand the contribution of an S-box to thwart differential and linear cryptanalysis. A focus is put of laying out *efficient* algorithms to compute both the Fourier transform and the Walsh transform.

2.2.6 Lattice reduction

A lattice is a discrete subgroup of $\mathbb{R}^n$. Many cryptographic schemes base their security on the hardness of lattice-related problems like finding a reduced basis a lattice (proven NP-complete) or finding the shortest vector of a lattice (proven NP-hard). The author presents here different approximation algorithms for those two problems of polynomial complexity such as the famous lattice reduction algorithm LLL.

2.2.7 Polynomial systems and Gröbner base computation

Many cryptanalytic task can be easily described by multivariate polynomial systems of equations so it's a fair question to ask how easy it is to solve those. This part deals with the most efficient tools to solve such system.

2.3 Applications

Four chapter are contained in this part each one of them applying the tools described in the previous part.

In the first chapter, the author analyzes LFSRs and the various attacks that can be mounted against cryptographic schemes based on one or several of them.

In the second chapter, the author presents some direct applications of lattice reduction such as Knapsack problems, finding the minimal polynomial of a real algebraic number, or attacking the NTRU public-key scheme. Is also presented the more intricate lattice-based attack on the RSA cryptosystem by Coppersmith.

The third chapter deals with elliptic curves and contains a detailed construction of the Weil pairing, a bilinear map on elliptic curves while the fourth and last chapter takes an in-depth look at index calculus-type algorithms with a focus on factoring and solving discrete logarithm and finite fields.

3 Opinion

Let's be clear, this book is not one that I would advise to an undergraduate that doesn't have a particular interest in cryptography : the first 20 pages go from defining confidentiality to the different flavors of authenticated encryption via zero-knowledge proofs and indifferentiability. Such

students are probably better off reading the already very complete "Cryptography : Theory and Practice" by Stinson or "Handbook of Applied Cryptography" by Menezes, Van Oorschot, and Vanstone.

That being said, in my opinion this book is a must-read/must-have-in-your-shelf for anybody seriously involved in the field of cryptography if only to give an overview of the range of techniques that can be applied to break cryptographic schemes and the cryptographic hurdles that ones needs to get over to design secure systems. Also Parts of the book can also easily be used as the basis for a cryptography course since every chapter contains exercises (hints and solution to some of them can be found on the author's website).

I particularly appreciated the focus on the practicality of the provided material : after an overview in plain English, every algorithm is clearly stated in the form of pseudo-code and many of them are also provided in C code. In that way the reader can easily follow the careful complexity analysis to convince himself that it is the algorithm he wants to use and then write an implementation of that algorithm from the book. The practical aspect of book shines as well through the effort of always presenting the most efficient algorithms for a given task with particular constraints (hardware, software, limited memory, limited computing power, ...) in mind.

To sum it up, this book is a mine of information on cryptanalysis and goes above and beyond to provide the reader with everything he needs to become a better cryptographer.

Review[3] of
Algorithmic Bioprocesses
Edited by Condon, Harel, Kok, Salomaa, Winfree
Springer, 2009
742 pages, hardcover

Review by
Aaron Sterling (sterling@iastate.edu)
Department of Computer Science, Iowa State University

1 Introduction

This book is a Festschrift that grew out of a workshop celebrating the 65^{th} birthday of Grzegorz Rozenberg. Among many other accomplishments, Rozenberg coined the phrase "Natural Computing," co-founded the journal of the same name, and also co-founded Theoretical Computer Science C (Theory of Natural Computing), and the Springer book series Natural Computing, of which this volume is a part. The editors describe their reasons for choosing the title "Algorithmic Bioprocesses": "the workshop did not cover the whole spectrum of natural computing research, but rather it was mostly focused on the interactions between computer science on the one hand and biology, chemistry, and DNA-oriented nanoscience on the other."

Algorithmic Bioprocesses contains 36 chapters, contributed by scientists from a wide variety of backgrounds. The first chapter, by Arto Salomaa, is a prose portrait of Rozenberg, both as a scientist and as a man. The remaining chapters are loosely grouped by biological subject area. Rather than attempt to summarize these chapters in order, I will assume that the reader of this review is a theoretical computer scientist with an interest in viewing biological processes through an algorithmic lens. I will describe some of the contributions, categorized by TCS area they seem most related to. Hopefully, this approach will help the reader identify research areas of potential interest.

2 Sampler of contents grouped by TCS area

2.1 Chemical Cost

Before considering chapters by more "formal" TCS area, I'll start by noting that one of the meta-questions arising from the 2007 NSF "Algorithmic Lens" workshops was how to quantify the "chemical cost" of nanomanufacturing [2]. Several contributions to *Algorithmic Bioprocesses* are provided by esteemed scientists with little CS background, who report on successes and obstacles from their laboratories. I will briefly consider two such chapters. The chapters in this vein may improve the ability of TCS researchers to create models that better address real-world laboratory complexities.

Nadrian Seeman is the inventor of DNA Nanotechnology. In 1998, he wrote a paper that offered rules of thumb for other experimentalists interested in DNA fabrication and DNA computation. His chapter, "The Perils of Polynucleotides Revisited," reconsiders these rules of thumb, and provides updates and annotations as necessary. As just one example, Seeman had encountered the problem

that DNA bases do not always pair as we are taught in high school (adenine to thymine, guanine to cytosine), but, rather, "Every base appears capable of pairing with every other base, including itself." While this is still a phenomenon to watch out for, some 21st-century nanochemists have exploited this to advantage (!), obtaining stable motifs with nonstandard base pairings.

In "Algorithmic Control: The Assembly and Operation of DNA Nanostructures and Molecular Machinery," Andrew Turberfield discusses the algorithmic steps laboratories go through to build two-dimensional DNA tiles and three-dimensional structures (such as tetrahedra), and also provides an introduction to the new subfield of molecular motors. There now exist DNA "tweezers," which open or close depending on the addition of components of DNA fuel, and DNA "walkers" that literally step along a track. Turberfield illustrates two such devices, and provides references to several more.

In what follows, I will consider chapters more directly related to TCS, ordered alphabetically by subject area.

2.2 Automata Theory

"Programmable DNA-Based Finite Automata" by Ratner and Keinan states: "Comparison between the Turing machine and the intracellular processing of DNA and RNA show remarkable similarities. Both systems process information stored in a string of symbols built upon a fixed alphabet, and both operate by moving step-by-step along those strings, modifying or adding symbols according to a given set of rules." Ratner and Keinan describe their real-world construction of a two-state, two-symbol finite automaton built from DNA, as well as their more recent DNA implementation of a three-state, three-symbol automaton.

The authors also address how to read the output of a biocomputation in real time. In one of their experiments, they cause the two-state, two-symbol machine to output differently-luminescing bacteria, depending on whether the answer is "yes" or "no." This literally means that the Petri dish turns blue if the input ab-sequence contains an even number of b's, and the Petri dish turns white if the input contains an odd number of b's. In a separate class of experiments, they cause the output to be produced in the form of a DNA strand. In their words, this demonstrates that "an appropriately designed computing machine can produce an output signal in the form of a specific biological function via direct interaction with living organisms."

Moving from the world of experiment to pure theory, J. Kari and Lukkarila apply a new reduction from cellular automata to Wang tiling in "Some Undecidable Properties for One-Dimensional Reversible Cellular Automata." A reversible cellular automaton (RCA) is one in which it is possible to track back uniquely from any configuration to the previous configuration. As natural computing processes are reversible (for example, steps in quantum computations are unitary matrices), RCA's are considered to be an interesting simplification of localized natural computation.

The Immortality Problem for Turing machines is, "Given this extended configuration (a tape that may have infinitely many non-blank symbols), will Turing machine T run forever?" (Answer: it's undecidable.) Motivated by this problem, the authors define notions of global and local immortality for two-dimensional RCA's. Roughly, an RCA is globally immortal if there is a subset of states (*e.g.*, the nonhalting states) such that a configuration exists so that all cells are in one of those states, and no cell at any future point will achieve a state outside that subset. Again roughly, an RCA is locally immortal if there is a starting configuration such that one cell never leaves the subset of desired states. The authors strengthen previous undecidability results about the domino

problem in Wang tiling, and use the new machinery to show that these immortality questions about RCA's are undecidable, both in general, and also for special subclasses of RCA's.

Last, but certainly not least, I will consider "Artificial Biochemistry" by Luca Cardelli. Cardelli's chapter is perhaps the most remarkable one in the whole volume, both for its ambition and its breadth of vision. Cardelli sets himself two tasks: creation of an automata theory that models the behavior of biological macromolecules, and creation of a concise visual notation for algorithmic biochemical interactions. About the second goal, he says: "Our main criterion is that, as in finite-state automata, we should be able to easily and separately *draw* the individual automata, both as a visual aid to design and analysis, and to emulate the illustration-based approach found in molecular biology textbooks. As a measure of success, in this paper, we draw a large number of examples."

About the first goal, Cardelli points out that standard models of automata do not capture the behavior of biological macromolecules: "A characteristic feature of biochemistry, and of proteins in particular, is that biological molecules can stick to each other to form *complexes*. They can later break up into the original components, with each molecule preserving its identity. This behavior can be represented by chemical reactions, but only by considering a complex as a brand new chemical species, thus losing the notion of molecular identity." (We have already implicitly encountered this modeling problem in Turberfield's chapter on nanomachines. TCS models of DNA self-assembly assume that, barring errors, the growing assembly is static, not dynamic; whereas DNA walkers attach to a substrate, then detach, then attach again further along the track.) Cardelli proposes: "In order to model the complexation features of biochemistry directly, we introduce *polyautomata*, which are automata that can form reversible complexes, in addition to interacting as usual."

Cardelli demonstrates that his formalisms allow for efficient construction of Boolean circuits, Boolean inverters, and toggle switches, among other constructions. It is too early to tell how influential his approach will be, but there is no question that he has identified key problems with current TCS formalisms, and has proposed intriguing possible solutions.

Before moving on, I should note that, even as finite automata and Turing machines are disappearing from CS curricula because of their lack of relevance to the "mainstream" computing market, small machines that can run small-but-powerful programs are more important than ever for nanocomputing. For example, Qian *et al.* recently designed a DNA implementation of a stack machine [4].

2.3 Bioinformatics

The chemical costs of Bioinformatics are laid out in "On the Concept of *Cis*-regulatory Information: From Sequence Motifs to Logic Functions" by Tarpine and Istrail. This chapter provides a near-slapstick exposition of challenges faced by experimentalists when they attempt to sequence genetic code. Problems include a multitude of different names for the same gene in the literature, the difficulty of distinguishing functional binding sites from nonfunctional ones, and the lack of clear criteria to test whether a model is correct. I finished this chapter thinking, "What a mess!" Bringing clarity to this mess, for example by defining useful metrics for correctness of a genomic model, seems like an intriguing research possibility.

Brijder and Hoogeboom bridge the gap between biological "messiness" and the crispness of graph theory in "Reality-and-Desire in Ciliates." They consider the reproduction of a ciliate whose cells contain two nuclei—a small one and a large one—instead of just one nucleus. As part of the

reproductive cycle, the genes in the small nucleus replicate themselves to form a large nucleus. Unlike many other reproductive cycles, no exterior organism is involved with this process, so fewer external variables are introduced. Hence, this process lends itself to graph-theoretic idealization, while remaining grounded in reality.

The authors provide a tutorial on the application of "sorting by reversal" to ciliate gene assembly. They begin, "In the theory of sorting by reversal one tries to determine the number of reversal operations to reorder ... a series of genomic 'blocks' from one species into that of another. An essential tool is the *breakpoint graph* (or reality-and desire diagram) which is used to capture both the present situation, the genome of the first species, and the desired situation, the genome of the second species." The chapter discusses applications of this breakpoint graph, states three theorems without proof, and concludes with two open problems.

The delightfully named "Monotony and Surprise," by Alberto Apostolico, is a purely mathematical overview of attempts to answer the question, "What makes a pattern inside a sequence interesting—and how can I find it?" One might think the least rigorous word in that sentence is "interesting," but Apostolico fixes "interesting" to mean "surprising," and considers increasingly sophisticated meanings of "pattern."

First, if a pattern is a finite string over alphabet Σ, its surprisingness within a sequence can be determined by the classical z-score (the departure of a pattern's actual frequency from its expected frequency). Now, suppose we don't know the string. Apostolico sketches an algorithm that finds overrepresented patterns in a sequence even though the patterns were not *a priori* defined. He also considers the problem of finding subsequences that are within a certain distance of a given pattern (for different notions of distance), and finding patterns with internal blanks (if abc is the pattern and $\bullet$ is the blank symbol, then abc, $a \bullet b \bullet c$, and $a \bullet\bullet\bullet b \bullet\bullet c$ are all sequences the algorithm should detect). Apostolico also addresses the metamathematical question, "When modeling a system, how can we determine which patterns we should look for?"

2.4 Combinatorics

The award for Most Mathematical Contribution goes to Hage and Harju for "On Involutions Arising from Graphs," because their motivation is one sentence long: "Involutions occur in biological processes especially in the base inversions of DNA." (Truthful joke: the first section of a computer science paper provides motivation, while the first word of a mathematics paper is "Let.") An involution of a group G is a mapping δ such that $\delta(\delta(g)) = g$ for all $g \in G$; the identity map and $(\forall g \in G)[\delta(g) = g^{-1}]$, the inversion of G, are simple examples of involutions. For a more visual example, let H be an undirected graph. Define H' to be a graph with the same vertex set, such that the edge ab is in the edge set of H' iff ab is not in the edge set of H. H and H' are said to be switch graphs, and the mathematical motivation for Hage and Harju's chapter is to investigate generalizations of switch graphs.

The authors provide an algorithm that finds all involutions of finite cyclic groups, and another algorithm to find all involutions of the direct product of two groups. They then consider the set of "skewed squares": for a group G and involution δ, the set of skewed squares is $\Delta^2(G, \delta) = \{a \in G \mid a = g\delta(g) \text{ for some } g \in G\}$. The authors show that if G is a direct product of two groups, the set of skewed squares of G is the direct product of the skewed squares of the two smaller groups, with appropriate involutions. (Note that there is a typo in the book in the definition of skewed squares. Δ^2 is defined for δ an inversion, but it should be for δ an involution.) This chapter concludes with

several open problems.

In keeping with the interdisciplinary spirit of the volume, Satoshi Kobayashi combines stochastic physics with enumerative combinatorics in "Applying Symmetric Enumeration Method to One-Dimensional Assembly of Rotatable Tiles." Kobayashi considers models of tile self-assembly in which the tiles can rotate. (By contrast, most, perhaps all, of the tile assembly models in papers at major TCS theory conferences make the simplifying assumption that tiles cannot rotate.) He then counts the number of possible configurations for certain kinds of tile assembly systems, and investigates the use of symmetry to limit the brute force counting required to answer questions about paths from one configuration to another. I found this chapter hard to follow, but it presents concepts that are important to consider when asking how we might make TCS models of self-assembly more realistic.

2.5 Distributed Computing

Perhaps the most theoretically satisfying chapter is "Programmability of Stochastic Chemical Reaction Networks" by M. Cook *et al.* A stochastic chemical reaction network (SCRN) is defined by a finite set of possible reactions, a finite set of chemical species at some initial concentrations, and a continuous-time Markov process according to which the system evolves. Chemists have used SCRNs for years, to describe systems. The authors of this chapter propose, instead, to take SCRNs as a primitive for a new programming paradigm: "while [SCRNs] are usually used *descriptively*, we will be using them *prescriptively*: we imagine that if we can specify a network of interest to us, we can then hand it off to a talented synthetic chemist or synthetic biologist who will design molecules that carry out each of the reactions."

I have placed this chapter in the Distributed Computing section because some of the results about SCRNs are independent rediscoveries of theorems about "population protocols," a model of distributed computing in which there are many agents of limited computational power [3]. The population protocol work is more abstract, so some of the results obtained there are more general. However, this chapter about SCRNs derives results within a specific model that is already in general use by practicing chemists, so it may have more real-world applicability. In any event, I believe more results lie available to the researcher who applies the literature of population protocols to SCRNs.

The proofs in this chapter are extremely pretty. As one example, to show that a certain question about SCRNs has the same computational complexity as the class of primitive recursive functions, the authors design a chemical system that computes sections (but not all!) of the Ackermann Function. The authors also relate SCRNs to Vector Addition Systems, Petri nets, register machines, and to Conway's number-theoretic programming language Fractran. TCS researchers may find these relationships to be useful entry points through which to say something about chemical computation.

"Process Calculi Abstractions for Biology" by Guerriero *et al.* provides a tutorial on how to model biological systems with process algebras (*aka* process calculi). Process algebras have been studied for years in concurrency theory, and, over the last five years or so, there have been many results in natural computing along the lines of, "Bioprocess X can be modeled by process algebra Y." (Also interesting: Cardelli's website contains an updated version of the "Artificial Biochemistry" chapter, which presents simulation results for some of his automata. The simulations are programmed in a process algebra language.) This line of research connects to the model checking

of biological systems (discussed below), as many model checkers accept as input distributed systems defined by process algebras.

Guerriero *et al.* exposit the process algebra approach with a running example from biology, the lymphocyte T helper, which, as they say, "is sufficiently complex to be an interesting case study for modeling issues, [but] abstract enough to allow us to omit a number of biological details." They survey several process calculi that have been developed to model biological systems, and discuss strengths and weaknesses of each. Some are better suited to model cellular systems; others, better suited to model signaling pathways. Overall, this research direction appears to still be in its infancy. The authors conclude with, "Process calculi for biology are in their pioneering phase and, although they are powerful and promising, a closer collaboration between life and computer scientists is required to bring appreciable results."

2.6 Membrane Computing/P-Systems

There are several chapters relating to Membrane Computing, or P-Systems, the computational objects of Membrane Computing. This is probably the area of least familiarity to most SIGACT News readers, so I will provide an informal definition of a P-System before summarizing one chapter in *Algorithmic Bioprocesses*.

First, though, a word of caution: like many other models of natural computation (including DNA computing, quantum computing and neural nets), there are claims that P-Systems can solve NP-complete problems in polynomial time. The Wikipedia page on P-Systems currently says, "It has been proven that certain P system variants are capable of solving the SAT (boolean satisfiability) problem in linear time," and provides a supporting reference to a journal article. Unless P=NP, this claim is overly optimistic. One variant of P-Systems achieves this special ability by being able to replicate membranes exponentially often in linear-many steps. This runs into the same obstacle encountered by the claims made in the 1990s that DNA computing could solve NP-complete problems efficiently: the computation requires an exponentially large amount of matter to perform. Even if one could transport matter to the computation site instantaneously (which one can't, so we're back to solving SAT in exponential time again), for problem instances of interesting size, one would need to compute with enough membranes that they would weigh more than the mass of the Earth itself. (As an aside, though I have nothing formal to offer here, my working hypothesis is that the class of problems efficiently solvable by DNA computing is *strictly weaker* than the class P, because DNA computations are charged a cost Turing machines are not: the physical space consumed by the computation. If a multi-tape Turing machine reaches a configuration in which its heads are separated by trillions of cells, the transition from that configuration to the next is still counted as one time step—no delay from communication between the heads. DNA molecules get no such free pass.)

A membrane system, or P-System (the "P" comes from the name of the originator, Gheorghe Păun; Păun and Pérez-Jiménez contributed a chapter to *Algorithmic Bioprocesses* about the modeling of brain function with membrane computing), is an unconventional model of computation that was biologically inspired, and now is being applied to computations about biology itself. A P-system includes a membrane structure—a set of membranes embedded in a unique all-inclusive membrane called the skin membrane. The exterior of the skin membrane is termed the environment. Objects—finite strings over a finite alphabet—are placed in each membrane, and in the environment, according to an initial state. Each membrane obeys evolution rules; different mem-

branes may follow the same rules, or different ones of their own. Like cellular automata, there is a globally synchronous evolutionary process, where at each time step, the objects in all membranes, and in the environment, are updated simultaneously, according to the rules that apply in each region. Wikipedia has articles on both P-Systems and Membrane Computing; there is also a web page (and a conference culture) dedicated to P-Systems [1].

I recommend "A Multi-Volume Approach to Stochastic Modeling with Membrane Systems" by Besozzi *et al.* because it provides an introduction to several methods of modeling stochastic systems, in addition to describing how the authors used a P-System with probabilistic evolutionary rules to model a genetic oscillator system within a cell. (Mizunuma and Hagiya's chapter, "Hybrid Method for Simulating Small-Number Molecular Systems," also provides a helpful introduction to issues of chemical modeling, from a different perspective.) Besozzi *et al.* discuss the performance of their algorithm during several different simulations, and compare it qualitatively to more standard methods of chemical simulation.

The overall sense I got from the membrane computing chapters is that this approach, and the process algebra modeling approach, are at similar levels of development. (Process algebras have the advantage of having been around far longer, outside of biology.) The use of membrane computing to model cellular processes is an intriguing idea that shows potential, but much more work is needed before it will be a powerful tool for biochemical practitioners.

2.7 Model Checking

I will conclude this section with "Quantitative Verification Techniques for Biological Processes" by Kwiatowska *et al.* A "biological pathway" can be thought of as a real-world algorithm executed by biological agents, such as the transformation of one compound into another by going through a series of steps. Biological pathways are often represented by diagrams reminiscent of flowcharts or directed graphs. However, while it is helpful to consider these pathways as sequential, the chemical systems that instantiate them are, in reality, stochastic, and can present challenges to model, as already discussed above.

Kwiatowska *et al.* provide a tutorial on the application of probabilistic model checking to the analysis of biological pathways with a running example from their own research, and by discussing related work. They show that, instead of obtaining just qualitative information about the behavior of a system when it is simulated multiple times, they can obtain the answers to quantitative questions through the exhaustive search of model checking, and posing queries in a probabilistic temporal logic such as, "What is the probability in the long run that there are precisely l kinases of this type activated?" or, "What is the expected number of reactions between these two agent types during the first t seconds?" In the words of the authors, "The intention is that probabilistic model checking should be used in conjunction with other, well-established approaches for analyzing pathways based on simulation and differential equations. In combination, these techniques can offer greater insight into the complex interactions present in biological pathways."

3 Opinion

In a real sense, this book answers the question, "What is the cutting edge of research connecting computer science with the biological sciences?" Of course there are subject areas not covered in the book (such as amorphous computing), but its breadth of content is impressive, and its combination

of advanced tutorials with ambitious new proposals is scientifically exciting. On the other hand, these same characteristics would not make it suitable for a textbook, except perhaps as a guide to an advanced graduate seminar. *Algorithmic Bioprocesses* will best serve TCS researchers who are looking for new questions to ask, and for new areas in which to apply their skills.

References

[1] The P-Systems web page. `http://ppage.psystems.eu/`.

[2] Arora, S., Blum, A., Schulman, L., Sinclair, A., and Vazirani, V. The computational worldview and the sciences: a report on two workshops. *NSF Report* (October 2007).

[3] Aspnes, J., and Ruppert, E. An introduction to population protocols. In *Middleware for Network Eccentric and Mobile Applications*, B. Garbinato, H. Miranda, and L. Rodrigues, Eds. Springer, 2009, pp. 97–120.

[4] Qian, L., Soloveichik, D., and Winfree, E. Efficient Turing-universal computation with DNA polymers. In *DNA 16* (2010).

Review of[4]
Vehicular Networks, from Theory to Practice
Edited by Stephan Olariu and Michele C. Weigle
CRC Press, 2009
472 pages, HARDCOVER

Review by
Yu Wang (yu.wang@uncc.edu)
University of North Carolina at Charlotte, Charlotte, NC 28223

1 Introduction

Vehicular ad hoc network (VANET) is an emerging new technology to integrate the capabilities of new generation wireless networks with smart vehicles. The idea is to provide (1) ubiquitous connectivity while on the road to mobile users, who are otherwise connected to the outside world through other networks at home or at the work place, and (2) efficient vehicle-to-vehicle (V2V) and/or vehicle-to-infrastructure (V2I) communications that enable the Intelligent Transportation Systems (ITS). ITS includes a variety of applications such as co-operative traffic monitoring, control of traffic flows, blind crossing (a crossing without light control), prevention of collisions, nearby information services, and real-time detour routes computation.

The idea of vehicular network has been proposed and studied for several decades. However, with the rapid development of wireless networking technology (such as mobile ad hoc networks), VANETs recently have drawn significant research interests from both academia and industry. Several major automobile manufacturers have already begun to invest in real inter-vehicle networks. For example, Audi, BMW, DaimlerChrysler, Fiat, Renault and Volkswagen have united to create a non-profit organization called Car2Car Communication Consortium (`http://www.car-to-car.org`) which is dedicated to increasing road traffic safety and efficiency by means of inter-vehicle communications. IEEE has also formed the IEEE 802.11p task group which focuses on providing wireless access for vehicular environment.

Even though vehicular network is just another form of wireless networks, it is distinguished from other kinds of wireless networks by its hybrid network architecture, special node movement characteristic, and new application scenarios. These unique characteristics pose many challenging research issues, such as mobility model, routing, data sharing, and security issues. This book edited by Olariu and Weigle aims to provide a comprehensive overview on current research in vehicular networks. It includes 14 chapters written by 33 researchers from both academia and industry. It covers a wide range of topics, including background knowledge from transportation science (e.g. traffic modeling), basic application scenarios of vehicular networks, current status of relevant projects in U.S. and Europe, and challenges and existing solutions on networking issues.

2 Summary

The book consists of 14 chapters which are organized into 6 sections covering topics from traffic engineering to human factors. The following sections briefly present the content of each section

and chapter.

2.1 Traffic Engineering

The first section focuses on the study of traffic data (on traffic flows or vehicle motions) in transportation systems. This data and its derived model are critical to design and operate an effective transportation system. The study will also benefit the design of vehicular networks. This section includes two chapters which cover two aspects of traffic engineering: monitoring and modeling.

Chapter 1: Traffic Monitoring provides an overview of the current state-of-the-practice in traffic monitoring. It starts with a discussion on the causes of congestion, and then reviews common types of traffic monitoring data (including traffic volume, vehicle classification, traffic speed and density, and travel time) and types of sensor technologies used by DOTs to gather information. Several applications of this traffic data are then introduced. Finally, four emerging probe-based methods for monitoring traffic are reviewed. These methods track the movements of a subset of the vehicle population in order to estimate the travel characteristics of all vehicles on the road.

Chapter 2: Models for Traffic Flow and Vehicle Motion focuses on mathematical descriptions of traffic dynamics. Different traffic models (including both models for longitudinal vehicle movement and models for discrete-choice situation) are introduced in this chapter. For example, the longitudinal movement models include car-following model, macroscopic traffic flow model, and intelligent driver model, while the discrete-choices models consider situations such as lance changes, turning decisions, and approaching a traffic light. These models can be used to optimize vehicular networks and evaluate and simulate various control methods in intelligent transportation systems. As one example, the authors of this chapter present how to simulate vehicle-to-vehicle communication using some of the traffic models.

2.2 U.S. and European Initiatives

The second section reviews current initiatives on vehicular communication systems in U.S. and European in two chapters respectively. The vehicular communication system (supporting both V2V and V2I communication) is called by different names in the two chapters: Vehicle-Infrastructure Cooperation (VIC) in U.S. and CAR-2-X Communication in European.

Chapter 3: Vehicle-Infrastructure Cooperation first reviews the developments of VIC in Europe, Japan and U.S. Then using the a testbed building in California (the VII California Testbed) as an example, this chapter describes the current VIC communication system of U.S. in detail (including roadside equipments, transport layer protocols, and performances). The chapter is concluded with a discussion of future trends of VIC.

Chapter 4: CAR-2-X Communication in Europe provides an overview of vehicular communication system (CAR-2-X system) from a European perspective. It first introduces the Intelligent Car Initiative supported by the European Union and reviews current relevant R&D projects conducted under this initiative. Then the authors present the CAR-2-X system architecture and discuss three key techniques in such system, namely, Geocast protocols, Internet integration, and frequency/channel allocation. Finally, an overview of standardization activities and an outlook of CAR-2-X communication in Europe are provided.

2.3 Applications

The third section covers a wide range of vehicular network applications in different levels of detail. Each of the chapters focuses on certain types or subtypes of applications.

Chapter 5: Safety-Related Vehicular Application gives an overview of safety-related applications. It starts with an introduction of basic message types and a message dispatcher used by safety applications. Then it outlines different types of safety applications: collision avoidance, public safety, sign extension, vehicle diagnostics/maintenance, and information from other vehicles. For each type of applications, a few examples are given. Then the authors explain two specific safety applications: Extended Emergency Brake Light (EEBL) and Cooperative Intersection collision Avoidance System (CICAS).

Chapter 6: Emerging Vehicular Applications summarizes vehicular P2P applications, such as interactive mobile game and distributed information sharing, in which certain content or data are shared among vehicles. This chapter classifies these applications into three categories based on the vehicle's role in managing data: source, consumer, or source/consumer. For each type of P2P applications, one or more examples are provided in detail. In the beginning of this chapter, the authors also give brief reviews of different wireless access methods, characteristics of VANET environment, and VANET routing protocols.

Chapter 7: Use of Infrastructure in VANETs focuses on how infrastructure is used for VANET applications. It starts with discussions on security and privacy in infrastructure-based VANETs which are the advantages provided by the road-side infrastructure. A few examples of VANET applications which use infrastructure are introduced. Then a large of portion of this chapter concentrates on detail of an infrastructure-based system (called NOTICE) which is developed by the authors of this chapter for the notification of traffic incidents and congestion. In NOTICE, sensor belts are embedded in the road at regular intervals and they act as the infrastructure to support inter-vehicle communication for notification messages.

Chapter 8: Content Delivery in Zero-Infrastructure VANETs focuses on a particular P2P application: content sharing in VANETs. After a brief review of previous work in VANET P2P area, the authors spend most of this chapter describing their solution: ZIPPER, a zero-infrastructure P2P system, where no infrastructure is used. ZIPPER is based on TAPR, a traffic-adaptive packet-relay protocol which is also proposed by the authors of this chapter. Both TAPR and ZIPPER are described in detail. Analytical and simulation results on the performance of ZIPPER are also provided.

2.4 Networking Issues

The fourth section concentrates on networking issues in VANETs, particularly, routing and localization. Routing aims to delivery packets from a source vehicle to a destination vehicle, while localization aims to obtain the location of an individual vehicle.

Chapter 9: Mobile Ad Hoc Routing in the Context of Vehicular Networks provides an overview on both mobile ad hoc routing and VANET routing protocols. It starts with a short summary of differences among different ad hoc networks, then it reviews most of the well-known ad hoc routing protocols based on classical classification: proactive/reactive protocols, hybrid protocols, and geographic protocols. Next, the authors describe the characteristics of VANETs and explain why VANET routing is more challenging than ad hoc routing. Several VANET-specific routing

protocols are then introduced in detail. They are categorized into three groups: source-routing-based protocols, geographical-routing-based protocols, and trajectory-based protocols.

Chapter 10: Delay-Tolerant Networks in VANETs also considers routing (or more general data dissemination which includes broadcast and multicast routing) in VANETs. However it models VANETs as Delay-Tolerant Networks (DTNs). In DTNs, continuous end-to-end connectivity is sporadic. This chapter first reviews the state-of-art of DTNs and routing protocols for DTNs (including both deterministic and stochastic methods). Treating VANETs as a special type of DTNs, this chapter introduces a classical call-following model as the underlying mobility model of VANETs. It then reviews a solution for vehicle-roadside data access problem where roadside units are used as an infrastructure in VANETs. Finally, for infrastructure-less VANETs, both sets of solutions for unicast routing and data dissemination are reviewed in detail.

Chapter 11: Localization in Vehicular Ad-Hoc Networks studies the localization problem in VANETs. Many VANET applications relies on the availability of position information of vehicles. This chapter first reviews three main groups of such applications. Then it introduces several localization techniques which can be used to obtain positions of vehicles (including GPS, map matching, dead reckoning, cellular localization, image/video processing, infrastructures localization services, and ad hoc localization). To achieve better accuracy and anywhere availability, multiple localization techniques can be combined. Data fusion techniques for this purpose are also introduced at the end of this chapter.

2.5 Simulations

The fifth section covers two simulation-related topics for vehicular networks: network simulators and mobility models used by simulators.

Chapter 12: Vehicular Mobility Models covers mobility models for vehicular networks. This chapter overlaps with Chapter 2 (traffic models), but with additional analysis/simulation results on these models and from networking perspective. It starts with a classification of vehicular mobility models: stochastic, traffic stream, car-following, and flows-interaction models. For each category, example models are given. Then it validates these mobility models by presenting several test results on both car-to-car interaction and flows interaction in simulation and comparing them with real-world behavior. Finally, the impact of different mobility models on the connectivity of a network is verified and discussed.

Chapter 13: Vehicular Network Simulators provides an overview of current vehicular network simulators. It starts with a summary of statistic of vehicular network simulators used by papers from ACM VANET workshop. Then it classifies all simulators into three categories: mobility simulators (for generating travel paths of vehicles, usually used with network simulators), general network simulators (supporting general networking simulation), and tightly integrated simulators (embedding the mobility simulator into the network simulator). For each category, several simulators are introduced.

2.6 Human Factors

The last section has only one chapter: *Chapter 14: Mental Workload and Driver Distraction with In-Vehicle Displays*, which discusses driver distraction from in-vehicle display. It covers (1) what's driver distraction and its impact, (2) how to study the effects of distraction, and (3) how to

design in-vehicle display to mitigate distraction. The topic of this chapter is irrelevant to vehicular networks.

3 Opinion

Overall, this book provides well-organized resources on vehicular networks and achieves the authors' goal: providing a broad and interdisciplinary overview of vehicular networks. Since the topic of vehicular network is relevantly new and this book is one of the first set of books covering this emerging networking topic, it is well-worthy to read for researchers or students who want to enter this new networking research area as an introduction handbook. However, for maturer researchers who already work in this area, it does not provide much new in-depth knowledge and insights. This book is also very usable as a textbook or reference book for a senior-undergraduate- or graduate-level CS networking course on vehicular networks. It does not assume that the reader has a great deal of background besides basic networking concepts. For teaching usage, the reader would definitely supplement the book with something more technical.

Even though this book has the word "theory" in its title, it covers very limited theoretical aspects of vehicular networking research except for two chapters on mobility/traffic models (Chapter 2 and Chapter 11) and a simple analysis of a protocol presented in Chapter 8. The majority of chapters are basically descriptions of various protocols, systems, or applications. Therefore, readers from theory community may not find exciting topics in this book.

As many edited handbooks, this book also has several obvious weaknesses: (1) several chapters overlap and share certain redundancy (such as traffic/mobility models and VANET routing); (2) the depth and level of details in each chapter are unbalanced among sections; (3) the last section on human factors is separated from the other chapters and its topic is completely out of scope of this book. In the reader's opinion, studying the impact of driver behavior on networking may be more suitable for this book than focusing on impact on driver from in-vehicle displays. Personally, the reader enjoys Section 4 and Section 5 the most (but maybe mainly because of his background).

Review of[5]
Graph Theory and Interconnection Networks
by Lih-Hsing Hsu and Cheng-Kuan Lin
CRC Press, 2009
706 pages, hardcover
Price on Amazon: new \$162.95**, used** \$97.09

Review by
Francesco Silvestri, silvest1@dei.unipd.it

1 Introduction

Interconnection networks appear in almost all systems where some components communicate: the telephone network and the communication system of a parallel/distributed computing platform are some notable examples. When designing the interconnection network of a system, many different issues should be taken into account, ranging from the physical properties of connections to communication protocols. One of the first problems to deal with is: which topology do we use for connecting nodes? Since an interconnection network can be seen as a graph where vertexes represent nodes and edges the links among them, graph theory has provided a valuable mathematical tool for the design and analysis of the topology of an interconnection network.

There are several ways to interconnect n nodes, and each one has its pros and cons. For example, a complete graph guarantees connectivity even if $n-2$ nodes or links break down, but it requires an high number $(n(n-1)/2)$ of links. On the other hand, a linear array requires only $n-1$ links, but just one fault disconnects the system. A number of graphs with interesting properties, regarding for example connectivity and fault tolerance, have been presented in the literature. The reviewed book *Graph Theory and Interconnection Networks* presents the structures and properties of some of these graphs and provides the background in graph theory for understanding them.

As the title suggests, the book is divided into two parts. The first one covers some classical notions and results in graph theory, such as graph coloring, matching and connectivity. The intent of this part is not to provide a profound coverage of graph theory, but just to put down necessary tools for studying the interconnection networks described in the following chapters. The second part of the book introduces some graph properties of interest in the design of interconnection networks, in particular connectivity, fault tolerance, Hamiltonian cycles, and diagnosability; these properties are also discussed for some graphs, like hypercubes, star and pancake graphs.

2 Summary

The book consists of 21 chapters which are partitioned into two parts. The first part (Chapters 1–10, pages 1–170) introduces some general notions and results in graph theory, while the second one (Chapters 11–21, pages 171–685) focuses on some graph properties regarding connectivity, fault tolerance, Hamiltonian cycles and diagnosability, and discusses them for some graphs.

Part I. *Chapter 1* is the mandatory chapter introducing the essential terminology, like graph, path and cycle. Some basic results, in particular on vertex degrees, are also given.

Chapters 2 builds on notions given in the previous chapter, and provides some examples of isomorphic graphs and of graphs containing a mesh or an hypercube even when k edges are removed (k-edge fault tolerant graphs).

Chapter 3 deals with the diameters of some graphs: shuffle-cubes, de Bruijn, star and pancake graphs. Few words are also devoted to routing algorithms for the aforementioned interconnections.

Chapter 4 is about trees and covers breadth and depth-first searches, tree traversals, number of (binary) trees.

Chapter 5 is devoted to Eulerian graphs. After describing the classical results, the chapter provides some applications, like the Chinese postman problem.

Chapter 6 is on graph matching and includes perfect matching as well as the more general notion of k-factor.

Chapter 7 deals with different measures of graph connectivity (e.g., k-connectivity, superconnectivity) and analyzes their relations; in particular, the relations between vertex cuts and the number of pairwise vertex-independent paths are shown (Menger's theorem).

Chapter 8 is about graph coloring, in particular vertex coloring: here we find bounds on the chromatic number and properties on color-critical graphs. The chapter closes with some remarks on edge coloring.

Chapter 9 covers Hamiltonian graphs. The chapter describes sufficient and necessary conditions of Hamiltonian graphs, the Hamiltonian-connectivity, and mutual independent Hamiltonian paths.

Chapter 10 closes the first part with some results on planar graphs.

Part II. *Chapter 11* deals with k-fault-tolerant Hamiltonian and k-fault-tolerant Hamiltonian-connected graphs. The first ones are graphs that contain an Hamiltonian circuit even with k edge or vertex faults; the second ones are graphs where each pair of vertexes is connected by an Hamiltonian path even with k faults. After describing some methods for constructing these kinds of graphs, some examples are investigated, including Peterson networks and pancake graphs.

Chapter 12 focuses on the special case of cubic 1-fault-tolerant Hamiltonian graphs. The chapter provides some construction schemes and discusses some graphs, like a variation of the Honeycomb mesh.

Chapter 13 covers k-fault-tolerant Hamiltonian-laceable graphs, that is, bipartite graphs where an Hamiltonian path exists between each pair of vertexes belonging to distinct sets even when k faults occur. This property is an extension of the aforementioned k-fault-tolerant Hamiltonian-connectivity which does not apply to bipartite graphs. As usual, after some general results, the chapter focuses on Hamiltonian-laceable graphs, like hypercubes and star graphs.

Chapter 14 covers k^*-connected graphs, which are graphs where each pair of vertexes is connected by k distinct paths which span all the vertexes (these paths compose a k^*-container of the vertex pair), and the extension to bipartite graphs, named k^*-laceable graphs, which requires the two vertexes to belong to distinct sets. These properties are discussed for some graphs, such as hypercubes and crossed cubes.

Chapter 15 focuses on cubic 3^*-connected and 3^*-laceable graphs, investigating their relations with 1-fault-tolerant Hamiltonian and Hamiltonian-laceable graphs.

Chapter 16 deals with the k^*-diameter of k^*-connected graphs. The length of a k^*-container is the length of its longest path, and the k^*-distance of a vertex pair is the minimum among the

lengths of all the k^*-containers associated with the pair. Then, the k^*-diameter is defined as the maximum k^*-distance between any two vertexes. The chapter investigates the k^*-diameter of star graphs and hypercubes.

Chapter 17 deals with pancyclic and panconnected graphs and their bipartite versions: a graph of n vertexes is pancyclic if it contains a cycle of length ℓ, for each ℓ from 3 to n; a graph is panconnected if between each pair of two distinct vertexes there exists a path of length ℓ, for each ℓ ranging from their distance to n. As usual, these properties are discussed for some interconnections, as hypercubes and augmented cubes.

Chapter 18 is devoted to k-mutually independent Hamiltonian graphs, which are graphs where k mutually independent Hamiltonian cycles start in each vertex of the graph. The chapter provides bounds on k for many graphs, like hypercubes, pancake and star graphs.

Chapter 19 deals with k-mutually independent Hamiltonian-connected graphs, that is, graphs where each pair of vertexes is connected by k mutually independent Hamiltonian paths. As in the previous chapter, bounds on k for different graphs are provided.

Chapter 20 is completely devoted to wrapped butterfly graphs and studies how the aforementioned properties apply to these graphs. Indeed, since a wrapped butterfly of n vertexes is bipartite if and only if n is even, it is preferable to discuss its topological properties in a unique chapter.

The conclusive *Chapter 21* covers the diagnosability of a network, that is, how the network can identify all the faulty nodes. The chapter describes two models for self-diagnosis, namely the Preparata, Metze and Chien model and the Maeng and Malek model. Then, some graphs are studied under both models.

3 Opinion

As evidenced by the Summary, *Graph Theory and Interconnection Networks* is almost devoted to advanced properties of some graphs representing interconnection networks. The first part provides only the basic background for handling the second one, and hence some graph theoretical topics may not be found in the book, such as Ramsey theory and random graphs (see, e.g., [1]). Also, the book does not cover, with few exceptions, classical interconnections like linear arrays and meshes, parallel and routing algorithms, performance evaluation and deadlocks (see, e.g., [3, 2]).

In my opinion, the main target of the book is a veteran researcher working on the topologies of interconnection networks. This reader may use this book as a reference where looking for results concerning the topological properties of some interconnections, such as hypercubes, butterflies, star and pancake graphs. The book may be also used by a researcher wishing to begin working on interconnection topologies since the first part provides the background required in the whole book. However, this reader may find some difficulties due to the lack of exercises and non-technical sections describing some concepts and results.

A remarkable feature of this book is the generous use of figures for explaining concepts and proofs. Furthermore, I also appreciate its completeness, since almost all of the statements are proved and suitable references to the literature are provided. Nevertheless, there are some aspects of the book that in my opinion should be improved in a future edition. The first one is that looking for a result in the book is not simple since the index is incomplete and there is no symbol index, which may be useful when you forget the meaning of a symbol. The second aspect is that there should be more non-technical sections with the purpose of describing at high level concepts and

results and of giving an overview of the content presented in a chapter: these sections may be useful for newbie researchers in interconnection networks.

References

[1] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2005.

[2] Jose Duato and Sudhakar Yalamanchili and Lionel M. Ni. *Interconnection Networks*. M. Kaufmann Publishers, 2002.

[3] Frank Thomson Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes*. M. Kaufmann Publishers, 1992.

Review[6] of

Transitions and Trees: An Introduction to Structural Operational Semantics
by Hans Hüttel
Cambridge University Press 2010, 272 pages
ISBN 978-0-521-19746-5, Hardcover ($99.00, £60.00)
ISBN 978-0-521-14709-5, Paperback ($50.00, £30.00)

Review by
Stephan Falke (falke@iti.uka.de)

1 Introduction

In order to reason about the behavior of computer programs it is mandatory to have a precise definition of the semantics (i.e., the meaning) of the various constructs that can be used in programming languages. A formal semantics of a programming language can serve as a standard for implementing compilers or interpreters and is necessary in order verify correctness of programs.

Several ways of providing the formal semantics of a programming language have been developed:

- *Denotational semantics*, pioneered by Dana Scott and Christopher Strachey in the 1960's, uses functions that assign a meaning to every construct in the programming language. Typically, these functions are state transformations operating on complete partial orders.
- *Structural operational semantics*, developed by Gordon Plotkin in the early 1980's, is based on transition systems that describe the evaluation steps of a program.
- *Axiomatic semantics*, due to Tony Hoare (1969), uses mathematical logic in order to define the semantics of the language constructs. For this, a set of rules describing the pre- and postconditions of the constructs is defined.
- *Algebraic semantics*, developed by Joseph Goguen *et al.* in the 1980's, is similar to denotational semantics. It also uses functions that assign a meaning to every construct in the programming language, but algebraic semantics is rooted in concepts from universal algebra.

As the subtitle suggests, the book *Transitions and Trees* deals almost exclusively with structural operational semantics. The final two chapters, however, present a brief introduction to denotational semantics, including an equivalence result between the denotational and the structural operational semantics of a basic imperative programming language.

2 Summary

The book consists of a preface, 15 chapters, 2 appendices, references, and an index. With the exception of the first chapter, each chapter contains exercises of varying difficulty.

Chapter 1: A question of semantics. This introductory chapter briefly discusses problems that arose due to the informal semantics given to early programming languages. Next, the four approaches to formal semantics outlined above are introduced. Finally, uses of formal semantics in the development of compilers and interpreters and in program verification and debugging are sketched.

Chapter 2: Mathematical preliminaries. This chapter reviews mathematical preliminaries including mathematical induction, logical notation, sets and operations on sets, relations, and functions.

Chapter 3: The basic principles. The third chapter starts the presentation of structural operational semantics (SOS). For this, it introduces the simple imperative programming language `Bims` using its abstract syntax. Since SOSs are based on transition systems, these are defined next. Typically, SOSs come in two flavors: big-step and small-step. In big-step SOSs, each configuration can only perform a single transition to its terminal configuration. In small-step SOSs, each transition describes a single step in a larger computation. This chapter presents big-step and small-step SOSs for (variable-free) arithmetic and Boolean expressions and discusses the issue of their determinacy.

Chapter 4: Basic imperative statements. The SOSs of `Bims` are completed in this chapter. Since statements in imperative programming languages alter the program state, this concept is introduced. Next, the SOSs of arithmetic and Boolean expressions are refined to allow variables. Furthermore, big- and small-step SOSs for statements are given. Finally, the equivalence of the big- and small-step SOSs is proved.

Chapter 5: Control structures. The only control structures in `Bims` are if-then-else statements and while-loops. This chapter enriches `Bims` by repeat-loops and for-loops. The concept of semantical equivalence is introduced and it is shown that these new control structures can already be simulated in `Bims`. Next, bounded nondeterminism is introduced and it is shown that its big-step SOS corresponds to angelic nondeterminism (non-terminating computations are suppressed) whereas its small-step SOS corresponds to demonic nondeterminism (non-terminating computations are not suppressed). Finally, non-communicating parallelism is introduced with a small-step SOS.

Chapter 6: Blocks and procedures (1). After considering the simple language `Bims` in chapters 3–5, this chapter introduces `Bip`, which extends `Bims` by blocks and parameter-free procedures. Since blocks entail a scope for variables, the simple program state used before needs to be replaced by an environment-store model. The various possible scope rules (fully dynamic, dynamic only for variables, dynamic only for procedure declarations, and fully static) give rise to four different big-step SOSs for `Bip`.

Chapter 7: Parameters. The procedures as introduced in chapter 6 do not accept parameters. This chapter extends `Bip` to `Bump`, where all procedures accept exactly one parameter. Furthermore, procedures may now be recursive, which was not allowed in `Bip`. Since there are several parameter passing mechanisms, big-step SOSs are given for call-by-value, call-by-reference, and call-by-name

parameter passing. The treatment of call-by-name requires the concept of α-conversion in order to avoid name clashes between local variables and variables passed in the parameter term.

Chapter 8: Concurrent communicating processes. Extending the non-communicating parallelism introduced in chapter 5, this chapter introduces communicating parallel processes. This is done using communication via channels, similar to the setting in process calculi such as CSP or CCS. Both synchronous and asynchronous communication is discussed. Next, the chapter gives a brief introduction to the π-calculus, where channel names may the communicated via channels. The equivalence of two semantics for the π-calculus is discussed.

Chapter 9: Structured declarations. The ninth chapter introduces structured declarations in the form of records, where records may contain both variables as well as procedure declarations. A big-step SOS for this extension of `Bip` is provided. Next, the dynamic creation of records in the form of objects is discussed, again by giving a big-step SOS.

Chapter 10: Blocks and procedures (2). The introduction of blocks and procedures in chapter 6 only considered a big-step SOS. This brief chapter introduces a small-step SOS for the same constructs, again considering different possible scope rules.

Chapter 11: Concurrent object-oriented languages. The eleventh chapter combines ideas from chapters 8 and 9 by introducing concurrently running objects that communicate using remote procedure calls. A small-step SOS for this setting is developed.

Chapter 12: Functional programming languages. Chapter 12 leaves the realm of imperative programming languages that has been the considered in chapters 2–11. After briefly recalling the defining features of functional programming and the history of functional programming languages, the λ-calculus is introduced as a theoretical foundation of functional programming languages. Finally, the simple (untyped) functional language `Flan` (a syntactic fragment of ML) is introduced and both big-step and small-step SOSs using call-by-value parameter passing are developed.

Chapter 13: Typed programming languages. All example programming languages introduced thus far are untyped. This chapter introduces types into the programing languages `Bump` from chapter 7 and `Flan` from chapter 12. It is discussed how to determine whether a given program is type correct and SOSs for the typed programming languages are developed. Furthermore, it is show that well-typed programs cannot go "wrong", i.e., do not produce type-related runtime errors.

Chapter 14: An introduction to denotational semantics. As the title suggests, this chapter is not about SOS. Instead, it gives a brief introduction to the ideas of denotational semantics. For this, the language `Bims` from chapter 4 is considered and a denotational semantics for arithmetic expressions, boolean expressions, and program statements is developed. A complication is caused by while-loops, whose denotational semantics is defined in terms of itself. The question of whether this definition is indeed well-defined is left for chapter 15.

Chapter 15: Recursive definitions. The final chapter of *Transitions and Trees* is concerned with recursive definitions. In order to ensure that these have a "least" solution, a fixed point existence theorem for continuous functions in complete partial orders is proved. This fixed point theorem is then used in order to show that the denotational semantics of `Bims` from chapter 14 is well-defined if the semantics of while-loops is taken as the least fixed point of the recursive definition. Furthermore, the equivalence of `Bim`'s big-step SOS and its denotational semantics is shown and further applications of the fixed point theorem are sketched.

Appendix A: A big-step semantics of Bips. This appendix collects all rules for the big-step SOS of `Bip` that were defined in chapters 3–6.

Appendix B: Implementing semantic definitions in SML. The second appendix briefly shows how to implement big-step and small-step SOSs of `Bims` in SML, resulting in a simple interpreter for `Bims`.

3 Opinion

Transitions and Trees is a detailed, rigorous, and thorough textbook on structural operational semantics on an advanced undergraduate level. It allows for some flexibility in teaching since there are several self-contained paths that can be taken through the book. The book's strength is the comprehensive coverage of many aspects of structural operational semantics. As the subtitle already suggests, alternative methods of providing the semantics of programming languages are only mentioned in passing (axiomatic semantics and algebraic semantics) or briefly introduced (denotational semantics). Textbooks that contrast the different approaches to programming language semantics include [1, 2]. *Transitions and Trees* is generally well written and the large number of examples and interspersed exercises help in understanding the material.

References

[1] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993,

[2] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-Verlag, 2007.

Review of[7]
Origins and Foundations of Computing
Author of Book: Friedrich L. Bauer[8]
Springer Verlag, 2010. ISBN 978-3-642-02991-2
Review by Haim Kilov haimk@acm.org

1 Introduction

Friedrich L. Bauer is one of the founding fathers of computing. He invented the stack (for the patent in German, see [1]), was the Chairman of the legendary NATO Garmisch Software Engineering Conference [2], and was one of the authors of the Algol 60 Report. Some of us may recall his outstanding book [3]. His current interests are in the history of computing and cryptology.

This book is a translation from the German edition *Kurze Geschichte der Informatik* published in 2007, and its target audience appears to be *the broader public* (from the very short review by Niklaus Wirth). In most but not all instances the contents is understandable to a relatively uninitiated reader.

2 Summary

This very concise book is not only, and not even mainly, about software (engineering) as it is understood now. Half of the book is about *calculations* and devices before 1935, and only 10 pages are about informatics after 1960. Bauer stresses that the roots of informatics *extend much further back into history* and notes the role of Leibniz who introduced the concept of *freeing humanity from the wearisome burden of monotonous mental activity*. Bauer goes even further into the past; for example, he mentions the Antikythera astronomical calculator of 82 BC (see [4] for a lot of very interesting details). Thus, we read a lot about the pre-history of modern informatics.

The breadth-first presentation is illustrated with 227 figures (mostly portraits and some devices). Pretty often, the reader encounters only somewhat commented upon pointers (names, concepts, terms) and has to look elsewhere for details. At the same time, the amount of material is huge, and the material is interesting and probably not always well-known.

The book starts with a Preamble consisting of two subsections: The Roots of Informatics, and Informatics and Mathematics. The next two sections are about numerical calculations and calculations using symbols (with a strong emphasis on cryptology). The following section *After 1890: In Thrall to Mechanical and Electromechanical Devices* is mostly about machines (including cryptological instruments), with digressions into analog computers, process control, etc. The *After 1935* section brings the reader to a more familiar territory of *Formal languages and algorithms*, *'Universal Machines'*, and *electronic solutions* (including a very terse overview of formal languages, computability, recursion and related numerous new ideas and techniques that began to flesh out the bare bones of informatics that revolutionized the ways of thinking and of working), while the final section *After 1960* shows how *Informatics begins to take shape* when one *could begin to suspect that software would one day acquire spectacular economic significance*. The Conclusion claims that

[7]
[8] In Cooperation with Heinz Nixdorf MuseumsForum. With editorial assistance from Norbert Ryska.

Informatics and microelectronics are mutually dependent. There is a huge index of names, an index of figures, and a reference list.

The importance and role of mathematics (*not classical mathematics but mathematics* as noted by Bauer in [2]) is emphasized throughout the book, and the reader is supposed to have some mathematical maturity. Bauer's statement

The supercilious sneers of socialites who take pride in saying "I was never any good at math" can, in the future, be countered with "If you had paid proper attention to math you might have made something of yourself."

is obviously pleasing. At the same time, his 1968 [!] statement that

systems should be built in levels and modules, which form a mathematical structure [2]

would perhaps be even more appropriate in this book, but it is not mentioned. Programming as a (mathematical) discipline in itself began to be discussed in the 1960s, both by the participants of the NATO Garmisch Conference and elsewhere (for example, independently in the then USSR by A. Brudno, A. Kronrod, B. Levi, and others). To quote N. Wirth, another software pioneer,

[i]t was slowly recognized that programming was a difficult task, and that mastering complex problems was nontrivial, even though—or perhaps because—computers were so powerful. Salvation was sought in "better" programming languages, in more "tools", and in automation. [...] [Computer system] complexity can be mastered intellectually by one tool only: abstraction. [5]

This ought to be of importance for the target audience of the book, but there is only one—very nice!—page there on software engineering, and the enormous intellectual challenges of informatics, noted in [2] and even earlier, for example, by E.W. Dijkstra in [6], are not stressed emphatically enough, although hints for the initiated are certainly there (*The candid [Garmisch] conference report did not fail to have a salutary effect on the development of informatics in the NATO countries and beyond.*). The deteriorating quality of *modern-day* software (and programming languages) is, unfortunately, not mentioned by Bauer in this book; for a strongly-written (historical) overview by N. Wirth, see [5].

3 Opinion

The book is very interesting, insightful, and quite enjoyable. Here and there, certain passages look like they were from Wikipedia provided that the pointers actually point somewhere for a curious reader to find out more. The prehistory of informatics is not too well-known, so its modern presentation is very welcome, and the emphasis on mathematics throughout is highly appropriate. For a reader well-versed in *A discipline of programming (E.W. Dijkstra)*, this book is an excellent historical overview; for a reader less well-versed in this discipline, Wirth's paper [5] appears to be an essential addition.

References

1. Broy, Manfred (ed.); Denert, Ernst (ed.) Software pioneers. Contributions to software engineering. Incl. 4 DVD. Berlin: Springer, 2002. 728 p.

2. Software Engineering. Report on a conference sponsored by the NATO Science Committee. Garmisch, Germany, 7th to 11th October 1968. Chairman: Professor Dr. F.L. Bauer. Co-chairmen: Professor L. Bolliet, Dr. H.J. Helms. Editors: Peter Naur and Brian Randell. 231 p.

3. Bauer, F.L.; Wossner, H. Algorithmic language and program development. In collaboration with

H. Partsch and P. Pepper. Transl. from the German. Texts and Monographs in Computer Science. Berlin-Heidelberg New York: Springer-Verlag, 1982. XVI, 497 p.
4. Russo, Lucio. The forgotten revolution. How science was born in 300 BC and why it had to be reborn. Transl. from the Italian by Silvio Levy. With the collaboration of the translator. Berlin: Springer, 2004. ix, 487 p.
5. N. Wirth. A brief history of software engineering. IEEE Annals of the History of Computing. July-September 2008, pp. 32-39.
6. E.W. Dijkstra, Programming considered as a human activity. EWD117 (1965). `http://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD117.html`.

Review of
Introduction to Scheduling [9]
Author: Yves Robert and Frederic Vivien
Publisher: Chapman & Hall-CRC 2010
978-1-4200-7273-0

Reviewer: Ben Fulton ben@benfulton.net

1 Overview

A wide variety of problems fall under the rubric of scheduling algorithms. An engineer attempting to select which job to run on a processor, a curriculum designer deciding which instructors are available for a semester and a shop manager trying to optimize an assembly line are all trying to solve problems with an underlying similarity: they all are attempting to determine the best way to allocate limited resources over time. Whether dealing with servers that break down, teachers going on vacation, or defective widgets on the line, each issue must be dealt with by analyzing times, availabilities, difficulties and other constraining parameters of the generic problem of resource allocation. In this book, authors Yves Robert and Frederic Vivien put together a series of essays on scheduling problems, solutions, and performance guarantees, from many of the top experts in the field.

2 Summary of Contents

Chapter 1 is devoted primarily to introducing the notation used to describe scheduling problems, a notation that can grow greatly complicated as the various parameters of scheduling problems are introduced. Several important definitions are introduced, such as makespans, feasibility, and precedence, and the three-field scheme of classifying scheduling problems according to their processor environment (one processor, many processors, identical processors or different), their job characteristics (whether jobs have deadlines or release times, or if some jobs must take place before others), and the schedule objective (getting all jobs done as quickly as possible, minimizing processor downtime, or minimizing lateness of the jobs). Different processor models are introduced, such as single-processor, parallel-processor, and multi-processor, and common scheduling terms such as general shop, job-shop and flow-shop problems are defined in terms of the three-field notation. Finally, a section is devoted to NP and the complexity of solving scheduling problems with various parameters.

Chapter 2 delves deep into approximation theory. The chapter mostly consists of theorems and proofs. Several polynomial-time algorithms are presented, some related to scheduling, some not, but only a very little space is used to explain the algorithms, with the majority of the chapter given over to proving the algorithms' complexity characteristics. In Chapter 3, Suzanne Albers considers problems that arise when scheduling jobs that arrive at unknown times and with unknown processing times. This is known as Online Scheduling. The analysis technique known as Competitive Analysis is introduced, and algorithms that attempt to minimize makespans are considered in its light: the classic List algorithm; Imbal, an algorithm which attempts to keep some machines lightly

loaded; and one nondeterministic algorithm. Also, minimizing flow time (the time between arrival and completion of a given job) is considered, and two algorithms to solve that problem are analyzed: processing the job with the shortest remaining processing time, and processing the job that has had the least time devoted to it.

Also in this chapter is an analysis of load-balancing using the Robin Hood algorithm, and then scheduling power-downs for idle machines. Finally, the case of variable-speed processors is considered. These processors are assumed to be able to increase speed and power together, and algorithms for minimizing power use over a series of jobs are considered.

Scheduling problems can have additional constraints beyond precedence requirements, deadlines, and due dates. In chapter 4, Uwe Schwiegelshohn considers weighted job scheduling, preemption, and release dates for a single processor. For multiple machines, both makespan optimization and completion time optimization are considered. (The authors make the point that while complex algorithms can be used to minimize worst-case problems, in practice simpler algorithms tend to be used to allow additional constraints. Therefore they focus attention on the simpler algorithms). For makespans, the algorithms of Longest Processing Time First and List Scheduling are considered. For completion time, the Shortest Processing Time algorithm is considered, and the rather complex proof of its efficiency is given.

Chapter 5 concerns scheduling of tasks that reoccur periodically, such as the manufacture of a complex part that requires several steps and needs many copies. This is known as cyclic scheduling. Author Claire Hanen begins by defining several concepts, including schedule throughput and schedule stability. A very general scheduling problem called a uniform task system is defined, and then refined to a periodic schedule. A polynomial algorithm for computing an optimal periodic schedule is demonstrated by mapping the schedule to a directed graph. Finally, periodic schedules with resource constraints are considered, and several interesting results are shown. A small section on using a priority rule to simplify the schedule concludes the chapter.

Chapter 6, although it is titled *Cyclic Scheduling for the Synthesis of Embedded Systems*, is actually about timed weighted event graphs (WEGs). After defining WEGs, definitions are given for precedence relations and for unitary WEGs, and for unitary WEGs, normalization and expansion routines are given. Finally, periodic schedules are defined and a polynomial algorithm for finding an optimal periodic schedule is given.

Algorithms used in partitioning parallel jobs across multiple machines, in the way that Google has implemented Map/Reduce, are the subject of chapter 7, "Steady-State Scheduling". Authors Olivier Beaumont and Loris Marchal begin by defining the problem, with a graph defining multiple machines and the connections between them, some applications that might be defined on the graph, and definitions of allocations and valid patterns. From there, an algorithm is defined to create a periodic schedule for the grid. Maximizing throughput is discussed, and some helpful polynomial algorithms explained.

Chapter 8, in many ways the most easily understandable chapter of the book, deals with divisible load scheduling, or algorithms for passing out bits of data to be computed in the style of the SETI@home project. The authors build to their subject by considering how to apportion data between several machines, both in bus-shaped and star-shaped networks. Several properties that an optimal solution must have are proved, and a polynomial-time formula then derived for optimal solutions. Finally, complicating factors such as latencies, multiple rounds of data, and return data are considered.

Chapter 9 concerns techniques for optimizing many objectives at the same time, such as

makespan and average time to completion, or sharing a processor fairly between multiple users. To solve these problems, definitions of optimal solutions are given, called Pareto-optimal solutions and zenith solutions. The complexity of finding such solutions is covered, and polynomial approximation algorithms for certain multi-objective problems are given. Finally, fairness is discussed, along with the difficulty of finding a universally satisfactory solution.

Chapter 10 demonstrates comparing the performance of systems by modeling with random incoming tasks. Several types of stochastic ordering are discussed, and then several properties of systems, both modeling a finite number of tasks, and with an infinite number of tasks, are shown to hold.

Finally, chapter 11 gives some models of network topology, and considers how the selected model may affect problems in scheduling. Examples are given for divisible load scheduling, and for efficiently redistributing data between processors.

3 Opinion

The title of the book is mildly misleading - an engineer wishing to solve a specific problem quickly should not pick up this book to learn about approaches to scheduling. It is an introduction to scheduling theory, not scheduling itself.

There are two large problems associated with scheduling even before a potential algorithm can be considered: recognizing the similarities between various problems, and designing notation to compactly describe those similarities. Robert and Vivien do a good job covering the existing notation, but the disjointed nature of the essay format might make it difficult for a student to relate a new problem to a problem in the book.

Although the level of difficulty of the chapters ranges widely, students should have at least exposure to an undergraduate algorithms class before attempting this book. The first chapter offers a quick review of "easy" and "hard" problems, but chapter 2 goes into some fairly complex proofs; on the other hand chapter 8, for example, should be accessible to most undergraduates. An astonishing amount of knowledge can be gleaned from this book, but many sections would probably be better left to graduate courses, and even graduate students might need a fair amount of explanation as they work through all the chapters. Still, the difficulty of the subject probably precludes any kind of lightweight treatment.

Review of[10]
Semantic Techniques in Quantum Computation
Edited by Simon Gay and Ian Mackie
Cambridge University Press, 2010
xiv+478 pages, ISBN: 978-0-521-51374-6 (Hardback, £60.00, $90.00)

Review by
Kyriakos N. Sgarbas (sgarbas@upatras.gr)
Electrical & Computer Engineering Department,
University of Patras, Hellas (Greece)

1 Overview

Although we still have to wait for a while (or a little bit longer) until we see quantum computers at our desktops, researchers are building logic systems, semantic structures, even programming languages for them. This is an edited volume containing 11 lengthy articles in the broad area of semantics for quantum computation. The articles are quite diverse, some are more technical, some are more theoretical, some are more review-oriented, and some are definitely research papers. After reading this review you might be interested in checking the site of the past QPL workshop[11] or the EPSRC Network on Semantics of Quantum Computation[12].

2 Summary of Contents

The book begins with a Table of Contents and a list with names, affiliations, and addresses of the 27 researchers who have contributed their articles. Then follows a 4-page Preface where the editors give a very brief overview of the evolution of the particular branch of quantum computation history since Feynman's original idea until the recent QPL workshops and the QNET research network on Semantics of Quantum Computation (in the framework of which most of the presented research was conducted); also a brief presentation of the articles loosely relating them into more specific thematic categories and the preface concludes with a one-page bibliography. The main content of the book is organized in 11 articles (each with its own bibliography) as follows:

Article 1 *"No-Cloning in Categorical Quantum Mechanics" by Samson Abramsky.* The no-cloning theorem is a fundamental theorem in quantum information processing stating that a qubit cannot be forced to copy an arbitrary unknown state of another qubit. The standard proof is rather straightforward, if we consider that all operations in quantum computing are unitary and reversible. The article relates the no-cloning theorem to *Joyal's lemma* (Lambek and Scott, 1986) in categorical logic and proposes an alternative proof of it from the perspective of the categorical formulation of quantum mechanics. It uses the graphical calculus for monoidal categories and proves that the existence of a qubit cloning operation is incompatible with the existence of the Bell states. (28 pages)

[10] ©2012, Kyriakos N. Sgarbas
[11] `http://web.comlab.ox.ac.uk/people/Bob.Coecke/QPL\_10.html`
[12] `http://www.informatics.sussex.ac.uk/users/im74/QNET/index.html`

Article 2 *"Classical and Quantum Structuralism" by Bob Coecke, Éric Oliver Paquette, and Dusko Pavlovic.* Graphical calculus is used again in this article to represent operations over classical and quantum states from a categorical semantics point of view. The analysis considers all options (i.e. deterministic, non-deterministic, and probabilistic operations for classical data, pure and mixed states for quantum systems) and describe the morphisms that control classical information flows between classical interfaces, as well as the information flow between the quantum universe and a classical interface. The article concludes with some open questions for future research. (41 pages)

Article 3 *"Generalized Proof-Nets for Compact Categories with Biproducts" by Ross Duncan.* This article presents the logic of compact closed categories and biproducts, which allows quantum processes to be represented as proof-nets. After an adequately detailed introduction to logic, processes, categories and their relation to quantum mechanics into the formation of appropriate proof-nets, follow the descriptions of the main categorical structures (i.e. monoidal, compact closed, freely constructed compact closed, compact symmetric polycategories, etc) up to the definition of zero objects and biproducts. The formal syntax of a tensor-sum logic is then introduced and used to produce tensor-sum proof-nets and generalized proof-nets constructed over the generators of compact symmetric polycategories. A normalization procedure is also defined for such proof-nets, based on local rewrite rules, resulting to the proof that the aforementioned proof-nets are strongly normalizing. (65 pages)

Article 4 *"Quantum Lambda Calculus" by Peter Selinger and Benoît Valiron*, as its title suggests, presents a typed lambda calculus for quantum computation. The corresponding operational and categorical semantics are defined, along with a type inference algorithm. The article examines some common quantum computation examples (namely, the Deutsch-Jozsa algorithm, quantum teleportation, and Bell's experiment) (Nielsen and Chuang, 2000) expressed in terms of higher-order functions and uses them as starting points in order to design the lambda calculus. There are adequate usage examples (including the quantum Fourier transform). The article concludes with a brief discussion on whether a concrete model of the quantum lambda calculus could be obtained. (38 pages)

Article 5 *"The Quantum IO Monad" by Thorsten Altenkirch and Alexander S. Green* presents a Haskell interface to quantum programming. After a brief example-based introduction to the basics of Haskell functional programming language, the functional interface to quantum programming is introduced as an analogy to the interface that the IO monad provides to conventional stateful programming. Some examples follow, showing how the quantum IO monad (Green, 2008) can be used to model Deutsch's algorithm, quantum teleportation, and quantum Fourier transform, building up to a complete implementation of Shor's algorithm. The article concludes with some implementation details concerning the quantum IO API. (33 pages)

Article 6 *"Abstract Interpretation Techniques for Quantum Computation" by Philippe Jorrand and Simon Perdrix* demonstrates that abstract interpretation (Cousot and Cousot, 1977) techniques can be successfully applied to quantum computing and exhibits how abstract interpretation can be used for establishing a hierarchy of quantum semantics and for analyzing entanglement evolution. After two very brief introductionary sections to abstract interpretation and quantum computing basics, a simple quantum imperative language (QIL) is defined and used for the expression of denotational semantics (probabilistic, observable, and admissible) and entanglement analysis. (29 pages)

Article 7 *"Extended Measurement Calculus" by Vincent Danos, Elham Kashefi, Prakash Panan-*

gaden, and Simon Perdrix presents an alternative quantum computation model called *Measurement - Based Quantum Computation* (MBQC). Instead of performing operations to the whole quantum register and save the measurement for the end as is the case in the traditional quantum circuit model, in MBQC one performs only single-qubit unitary operations and selective intermediate measurements to direct the computation. In the article the authors describe the syntax, the operational and denotational semantics of the MBQC model, prove its universality and present an algebra on MBQC *patterns* (i.e. the series of MBQC operations that produce the intended computation) with corresponding examples. Using this algebra, they manage to produce standardized versions of MBQC patterns that although semantically equivalent to the initial ones they are computationally more efficient. The authors also provide methods for converting patterns to quantum circuits and vice versa. (76 pages)

Article 8 *"Predicate Transformer Semantics of Quantum Programs" by Mingsheng Ying, Runyao Duan, Yuan Feng, and Zhengfeng Ji.* The article begins with a review of the state transformer semantics of quantum programs represented via superoperators as suggested by Selinger (2004), and a review of D'Hondt-Panangaden's (2006) theory of quantum weakest preconditions. Then it focuses on projection operators and presents relevant predicate transformer semantics using Birkoff-von Neumann (1936) quantum logic, also introducing various healthiness conditions for quantum programs. The authors examine the relationship between quantum predicate transformer semantics and projective predicate transformer semantics as introduced in the article. Finally, quantum versions of the universal conjunctivity and termination laws are proved, along with a generalization of Hoare's (1971) induction rule. (50 pages)

Article 9 *"The Structure of Partial Isometries" by Peter Hines and Samuel L. Braunstein* uses *partial isometries* (Halmos and McLaughlin, 1963) to study the dynamic processes (as unitary maps) and measurements (as projectors) involved in quantum computation, from order-theoretic and category-theoretic viewpoints. The article proves that the subspace ordering of projectors on a Hilbert space is a special case of a partial isometry ordering. Using this partial ordering the authors define a category of partial isometries (which is proved to be an *inverse category*) and compare it to compact closed categories and monoidal closed categories in order to exhibit an inherent incompatibility between the "categorical foundations" and "orthomodular lattices" approaches to the foundations of quantum mechanics. (28 pages)

Article 10 *"Temporal Logics for Reasoning about Quantum Systems" by Paulo Mateus, Jaime Ramos, Amílcar Sernadas, and Christina Sernadas* uses as a starting point a restricted sublogic of the exogenous logic dEQPL (Mateus and Sernadas, 2006) for reasoning about quantum states and combines it with computational tree logic (CTL) (Clarke and Emerson, 1981) and linear temporal logic (LTL) (Pnueli, 1977). The authors provide axiomatization, a weakly complete Hilbert calculus, as well as SAT and model-checking algorithms. (25 pages)

Article 11 *"Specification and Verification of Quantum Protocols" by Simon J. Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou* presents QMC (Quantum Model Checker), an automated verification tool utilizing model-checking techniques for protocols used in quantum information and quantum cryptography applications. QMC uses an imperative concurrent specification language for quantum protocols (QMCLang) which is also defined in the article. The authors provide many implementation details and present applications to several case studies, including quantum teleportation, quantum coin-flipping, quantum key distribution, and quantum error correction. (59 pages)

The book concludes with a 6 page Index covering all articles.

3 Opinion

I have to confess that I am generally not very fond of edited volumes; in fact, I never liked the policy of collecting a set of papers and publish them as a book. But reading this one, I was pleased to find a nice exception. The articles are really good, well written, from leading researchers in their fields, and they provide state-of-the-art research information.

I refer to them as "articles" instead of "chapters" because they are exactly that, independent articles (review and/or research papers) that one can read selectively, with any order. They do not cross reference one another even when they discuss the same notions. However, in the Preface the editors group articles 1-3 within the category-theoretic framework for quantum mechanics, articles 4-8 into applications of semantic techniques, and articles 9-11 on quantum logic.

Most of the articles require a substantial theoretical background to be understood, but this background seems rather imbalanced. Concerning semantics, the reader should be familiar with many different aspects (you may check the content descriptions of the articles to realize the range), but concerning quantum computation just the classic textbook (Nielsen and Chuang, 2000) is more than enough in most cases. Therefore, the overall impression is of a set of articles trying to extend existing semantic techniques to quantum computers, rather than examining quantum computers and building special semantic techniques for them.

In conclusion, this is a well written and interesting research oriented book. I write "research oriented" because I could not imagine using it in class as a textbook, although some of the articles might interest some post-graduate students. But indeed this book provides a great source of information for all researchers working in the fields of logic, semantics, and quantum computation.

References

1. Birkhoff, G., and von Neumann, J. (1936) The Logic of Quantum Mechanics. Annals of Mathematics, vol.37, pp.823-843.

2. Clarke, E. M., and Emerson, E. A. (1981) Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logics. In Proceedings of the Workshop on Logics of Programs, LNCS 131, Springer-Verlag.

3. Cousot, P., and Cousot, R. (1977) Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction of Approximation of Fixpoints. In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp.238-252, ACM Press.

4. D'Hondt, E., and Papangaden, P. (2006) Quantum Weakest Preconditions. Mathematical Structures in Computer Science, vol.16, pp.429-451.

5. Green, A. (2008) The Quantum IO Monad, source code and examples. `http://www.cs.nott.ac.uk/~asg/QIO/`.

6. Halmos, P. R., and McLaughlin, J. E. (1963) Partial Isometries. Pacific Journal of Mathematics, vol.13, no.2, pp.585-596.

7. Hoare, C. A. R. (1971) Procedures and Parameters: An Axiomatic Approach. In Engeler, E. (ed), Symposium on Semantics of Algorithmic Languages, pp.102-116, Lecture Notes in Mathematics 188, Springer Verlag.

8. Lambeck, J., and Scott, P. J. (1986) Introduction to Higher-Order Categorical Logic. Cambridge University Press.

9. Mateus, P., and Sernadas, A. (2006) Weakly Complete Axiomatization of Exogenous Quantum Propositional Logic. Information and Computation, vol.204, no.5, pp.771-794.

10. Nielsen, M.A., and Chuang, I. L. (2000) Quantum Computation and Quantum Information. Cambridge University Press.

11. Pnueli, A. (1977) The Temporal Logic of Programs. In Symposium on the Foundations of Computer Science (FOCS), IEEE Computer Society Press, Providence, Rhode Island, pp.46-57.

12. Selinger, P. (2004) Towards a Quantum Programming Language. Mathematical Structures in Computer Science, vol.14, pp.527-586.

Review of[13]
Modern Computer Arithmetic
by Richard Brent and Paul Zimmermann
Cambridge University Press, 2010
xvi+ 221 pages, HARDCOVER

Review by
Song Yan
syan@math.harvard.edu

1 Introduction

Fast algorithms for arithmetic on modern computers are extremely important in many areas of mathematics, computer science, and especially cryptography. This book is about algorithms for performing arithmetic, and their implementation on modern computers. More specifically, it collects, describes and analyzes state-of-the-art algorithms for arbitrary precision arithmetic (integers, integers modulo n, and floating-point numbers). It is an excellent addition and timely update to many of the well-known books and references in the field, such as those by Aho, Hopcroft and Ullman [2], Borodin and Munro [3], Brent [4], von zur Gathen and Gerhard [6], and particularly Knuth [7].

2 Summary

The book consists of the following five main chapters:

Chapter 1 (46 pages) describes integer arithmetic, including representation, addition, subtraction, multiplication, division, roots, gcd, and base conversion, etc. Readers who are familiar with polynomial arithmetic will find that many algorithms for polynomial arithmetic are similar to the corresponding algorithms for integer arithmetic discussed in this chapter.

Chapter 2 (32 pages) deals with modular arithmetic, including representation, multiplication, division/inversion, exponentiation, conversion, applications of FFT, and the use of the Chinese Remainder Theorem. Modular (some times called clock) arithmetic is a system of arithmetic for integers, where numbers wrap around upon reaching a fixed value, the modulus, say, e.g., 7 in the case of the weekdays.

Chapter 3 (45 pages) discusses the basic arithmetic operations such as addition, subtraction, comparison, multiplication, division, square root, algebraic functions, and conversion on arbitrary precision floating-point numbers, similar to the arithmetic operations on arbitrary precision integers discussed in Chapter 1. The algorithms in this chapter focus on correct rounding, extending the IEEE 754 standard in a natural way to arbitrary precision.

Chapter 4 (60 pages) considers various applications of Newton's method and its variants to the computation in arbitrary precision of functions such as sqrt, exp, ln, sin, cos, and more generally functions defined by power series or continued fractions. As the computation of special functions

is a huge topic, so the algorithms presented in this chapter are selective and at a rather high-level, omitting the detailed analysis.

Finally, Chapter 5 (6 pages) gives pointers to software tools, implementations, useful web-sites, mailing-lists, and on-line documents.

At the end of the book, there are 235 related bibliographic entries and more than 1000 index entries. There is also one page summary of the complexities of integer, modular and floating-point arithmetic operations.

The book also contains many helpful exercises, varying considerably in difficulty; some of them may be suitable as student projects.

Many algorithms from the book are implemented in the GNU MP and MPFR libraries.

Errata for the book may be found at `http://maths.anu.edu.au/{\char'176}brent/pd/CUP-errata.txt`.

3 Opinion

This is a concise, well-written and beautiful book in modern computer arithmetic. I found the book very pleasant to read. Compared with other books in the field, the book have the following unique and important features:

1. It gives detailed algorithms for all operations (not just multiplication as in many textbooks), such as addition, subtraction, comparison, division, roots, gcd, base conversion, exponentiation, etc.

2. It gives detailed algorithms for all size ranges (not just schoolbook methods or FFT-based methods). For example, in addition to the schoolbook $\mathcal{O}(n^2)$ method and Schönhage-Strassen FFT-based $\mathcal{O}(n \log n \log\log n)$ method, the book also includes Karatsuba and Toom-Cook methods for an intermediate range of precisions. It also considers the case where the operations have different lengths (not just the same length), and the methods that are not FFT-based, such as the divide-and-conquer methods, the Newton's method and the Toom-Cook method.

3. It gives detailed algorithms for computation of floating-point numbers ([1] and [5]) and for evaluation of special functions.

4. It gives detailed algorithms for polynomials, as they are simpler due to lack of carries.

The book should be useful for graduate students and final-year undergraduate students in computer science and mathematics (although it is not intended to be used as a textbook), researchers in discrete and computational mathematics, computer algebra, number theory, and cryptography, and developers of arbitrary precision libraries. It should be particularly useful if readers can read the book in conjunction with Knuth's classic book [7].

References

[1] ARITH Proceedings, *Proceedings of the IEEE Symposium on Computer Arithmetic 1969–2011*. `http://www.acsel-lab.com/arithmetic/html/`.

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier Computer Science Library, 1975.

[4] R.P. Brent, *Algorithms for Minimization without Derivatives*, Dover Publications, 2002. (Reprint from Prentice-Hall, 1973)

[5] N. Brisebarre and F. de Dinechin, et al. *Handbook of Floating-Point Arithmetic*, Birkhäuser, 2009.

[6] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 2nd Edition, Cambridge University Press, 2003.

[7] D. E. Knuth, *The Art of Computer Programming II: Seminumerical Algorithms*, 3rd Edition, Addison-Wesley, 1998.

Review of[14] of
Design of Approximation Algorithms
by David P. Williamson and David B. Shmoys
Cambridge, 2011
516 pages, Hardcover

Review by
Deeparnab Chakrabarty, deeparnab@gmail.com

1 Introduction

Many optimization problems are NP-hard, and unless P = NP there do not exist polynomial time algorithms to solve them exactly. One way to cope with this hardness is to design *approximation* algorithms, which run in polynomial time and return solutions guaranteed to be within a certain factor of the actual optimum solution. For instance, c-approximation algorithm for the traveling salesman problem takes as input an instance, and finds a tour whose cost is *provably* within c times that of the optimal tour. The quality of the algorithm is governed by how small the factor c is.

The study of approximation algorithm began about three and a half decades back, almost as soon as Karp's famous paper on 21 NP-complete problems appeared. Since then the field has seen a tremendous growth in its set of results and techniques, and applicability. In this book, Williamson and Shmoys provide an in-depth coverage of the various techniques developed over the years, as well as expositing on state-of-the-art results.

2 Summary

In the opening chapter of the book, the authors start off with a discussion on the importance of approximation algorithms. Subsequently in the chapter, the authors focus on one problem, the *set cover problem*, and discuss four different techniques of designing approximation algorithms for it. This sets up the tone of the book: the main focus of the book seems to be the techniques rather than the problems, although these almost always go hand-in-hand. In the book, the authors discuss seven different techniques, and each technique has two chapters devoted to it. The first introduces the technique via relatively straightforward applications, while the second, which starts with the prefix "Further uses of ... ", discusses more involved ones. The authors also provide one chapter on inapproximability, the study of what *can't* be done, and conclude with ten open problems which they believe will shape the field further in the years to come. Below, I provide short descriptions of these techniques; the readers should look into the book for a much more enlightening discussion.

- *Greedy and Local Search Algorithms.* Arguably, these are the first techniques algorithm designers try out when faced with a problem: from a given solution can one improve the solution by making best possible moves (greedy), or moves that improve upon the current scenario (local search). Many a times such a simple minded procedure does give provably good algorithms. The authors exhibit this for various problems: the traveling salesman problem and

[14]

the parallel machine job scheduling problem being two notable ones in the first chapter. In the second chapter on this topic, among other things the authors describe the local search algorithm for the k-median problem, which till date is the best algorithm known for it.

- *Rounding Data and Dynamic Programming.* In this chapter, the authors describe the use dynamic programming in the design of approximation algorithms. One of the ways to do is to "group" the input data into a small number of classes, and then exploit the structure in the problem to run a dynamic program. The number of classes need to be small enough so that the algorithm runs in polynomial time. Of course, this coarsening of data loses information, and the solution obtained is only an approximate one. This procedure often leads to the design of polynomial time approximation schemes (PTAS), algorithms which can get a solution within an $(1+\epsilon)$ factor and run in time polynomial in n when ϵ is a constant. The authors describe this for the knapsack and bin packing problem in the first chapter, and in the second they describe the algorithm for the Euclidean traveling salesman problem.

A simple but powerful idea in the design of approximation algorithms is to search for *polynomial time* computable (lower/upper) bounds on the optimum solution. One way to do so is to cast the problem as integer linear/quadratic programs, and using the linear semidefinite programming relaxations to obtain the bounds.

- *Deterministic Rounding of Linear Programs.* When a problem is cast as an integer linear program, one often uses variables which are supposed to take values in $\{0, 1\}$ where the value 1 indicates whether the corresponding element is picked in the solution or not. The relaxation to a linear program returns solutions where the variables take values in the continuous range $[0, 1]$. One class of approximation algorithms takes such a fractional solution and "rounds" it to a feasible integer solution. In the first chapter on rounding algorithms, the authors describe deterministic algorithms for the uncapacitated facility location problem and the bin packing problem, among others. In the later more detailed chapter, the authors describe more sophisticated rounding algorithms for the generalized assignment problem, and also introduces the strong technique of *iterative rounding*. In the latter, the algorithm proceeds in iterations rounding one variable per iteration – this strong technique has in the past five years led to a flurry of improved approximation algorithms for various problems.

- *Randomized Rounding of Linear Programs.* As mentioned above, the solution to the linear programming relaxation returns variables in the continuous range $[0, 1]$. One way to interpret these fractional quantities as probabilities that the corresponding element is present in the solution or not. Using this, several *randomized* approximation algorithms have been proposed. In the first chapter, the authors describe algorithms for many such problems including integer multicommodity flow problem, the problem with which this technique arguably arose. In the second chapter on randomized rounding, the authors describe improved algorithms for the facility location problem, and also touch upon some very recent developments on the Steiner tree problem. In these two chapters, the authors also describe the technique of *random sampling* in approximation algorithm design, a technique which works provably well when the input instances are dense.

- *Randomized Rounding of Semidefinite Programs.* Optimization problems can often be cast as integer *quadratic* programs, which can be relaxed to semidefinite programs. Solutions

to semidefinite programs return *vector* variables in high-dimension corresponding to each element. The rounding algorithm must convert such a vector solution into a binary solution of whether the element is picked or not. The authors describe the seminal work of Goemans and Williamson who started off this technique with their approximation algorithm for the max-cut problem. In the subsequent chapter on semidefinite programming, the authors describe rounding algorithms for the coloring problem, and for the *unique games* problem. The latter is, as of today, one of the central problems in theoretical due to an eponymous conjecture.

- *The Primal-Dual Method.* (Almost) every linear/semidefinite program has a dual linear/semidefinite program which is of the "opposite type" and has the same value. For instance, the dual of a minimization linear program is a maximization one, and the value of any feasible solution to the dual is a lower bound on the optimum solution of the primal. Primal-dual algorithms exploit this by constructing a solution to the problem at hand *and* a dual solution, and prove guarantees by comparing the costs of the two. Often times, the two solutions are constructed simultaneously, and the algorithms are indeed intertwined. Such algorithms are desirable since these almost never actually solve the linear programs (often an time-intensive step). The authors describe this technique using basic problems such as the shortest path problem, the facility location and the k-median problem. In the later section, the authors describe the primal-dual algorithm for the prize collecting Steiner tree problem.

- The seventh technique described in the book is a strong technique based on the theory of metric embeddings. Many optimization problems, such as the traveling salesman problem, have an underlying metric space on which they are defined. The theory of metric embeddings considers the problem of embedding one class of metric into another while minimizing the 'distortion' in the distances. There are two ways in which this becomes relevant to approximation algorithms. Firstly, the second metric might be 'simpler', and designing algorithms on such metrics may be easier. Then the distortion straightaway relates to the approximation factor. Secondly, it is known that many partitioning problems are "equivalent" to how well one metric embeds into another. In this spirit, a lot of research has gone into this area in the past decade. The authors start of by describing algorithms for various cut-problems (minimum cut, multiway cut, multicut), and algorithms for embedding general metrics onto tree metrics. In the second chapter on cuts and metrics, the authors describe the recent breakthrough on the sparsest cut problem. This description is probably the longest and most technically challenging part of the whole book.

3 Opinion

Any researcher interested in approximation algorithms would benefit greatly from this new book by Williamson and Shmoys. It is an ideal starting point for the fresh graduate student, as well as an excellent reference for the experts in the field. The first part of the book (which introduces the various techniques) is accessible to maybe a senior undergraduate, although many chapters in the latter half may require more mathematical maturity. The writing style is very clear and lucid, and it was a pleasure reading and reviewing this book.