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. **Set Theory For Computing: From Decision Procedures to Declarative Programming with Sets** by Domenico Cantone, Eugenio Omodeo and Alberto Policriti. Reviewed by Robert J. Irwin. This book is about computable set theory. By looking at set theory with a view towards computation you obtain some problems of interest, some solutions, and some insights into computing.

2. **Theory of Computational Complexity** by Ding-Zhu Du and Ker-I Ko. Reviewed by E. W. Čenek. This is a text for Computational Complexity that covers uniform, non-uniform, and probabilistic computation. There is alot of material in this one.

3. **Computer Arithmetic Algorithms: A Review** by Israel Koren. Reviewed by George A. Constantinides This is a textbook for advanced undergraduates or beginning grad students on the hardware underlying arithmetic algorithms.

4. **An Introduction to Quantum Computing Algorithms** by Arthur O. Pittenger. Reviewed by Andrea Marchini. This is a grad text in Quantum Computing.

Books I want Reviewed

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

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

Books on Algorithms, Combinatorics, and Related Fields

1. *Algorithms: Design Techniques and Analysis* by Alsuwaiyel.

2. *Genomic Perl: From Bioinformatics Basics to Working Code* by Rex Dwyer.

3. *Immunocomputing: Principles and Appliations* by Tarakanov, Skormin, Sokolova.

4. *Diophantine Equations and Power Integral Bases* by Gaal.

5. *Solving Polynomial Equation Systems I: The Kronecker-Duval Philosophy* by Teo Mora.

6. *Computational Line Geometry* by Pottmann and Wallner.

7. *Linear Optimization and Extensions: Problems and Solutions* by Alevras and Padberg.

8. *The Design and Analysis of Algorithms* by Levitin.

9. *Data Structures, Near Neighbor Searches, and Methodoolgy: Fifth and Sixth DIMACS Implementation Challenges* Edited by Michael Goldwasser, David Johnson, Catherine McGeoch.

Books on Cryptography and Books on Learning

1. *RSA and Public-Key Cryptography* by Richard Mollin.
2. *Data Privacy and Security* by David Salomon.
3. *Elliptic Curves in Crytography* by Blake, Seroussi, and Smart.
4. *Cryptanalysis of Number Theoretic Ciphers* by Wagstaff.

Books on Complexity and Logic

1. *Essentials of Constraint Programming* by Fruhwirth and Abdennadher
2. *Boolean Functions and Computation Models* by Clote and Kranakis.
3. *Modal Logic* by Blackburn, de Rijke, and Venema.
4. *Term Rewriting Systems* by Terese.

Review of

SET THEORY FOR COMPUTING: From Decision Procedures to Declarative Programming with Sets[2]

Authors: Domenico Cantone, Eugenio Omodeo and Alberto Policriti

Series: Monographs in Computer Science

Springer-Verlag, 2001

Hardcover, xviii + 409 pages, $51.50 from the publisher

Reviewer: Robert J. Irwin (rjirwin@cs.owego.edu)

1 Overview

Sets are the basic entities from which the familiar objects of mathematics are constructed. Set theory provides the basis for reasoning about such objects. The foundational importance of sets, what they are and what can be said about them (roughly, their semantics and syntax), accounts for the historically entangled lines of development of set theory and logic. Full axiomatic set theory, in any of its standard variants such as the Zermelo-Fraenkel theory (ZF), is first-order predicate calculus together with a small number of axioms or axiom schemes. The theory is astonishingly compact for its power and expressiveness. One drawback of its tremendous expressive strength, as established by Gödel's first incompleteness theorem, is that the theory is undecidable.

All usable programming languages provide aggregate objects of a sort that can be bent to the representation of sets, or at least finite sets. For example, AWK, Javascript, Perl and SNOBOL, among others, all furnish associative arrays, under various rubrics. Functional programming languages such as Haskell, ML and Scheme offer lists. Object-oriented languages like Java and C++ provide standard "container classes". None of the commonly used programming languages support sets with anything like the economy and generality of the usual mathematical syntax and semantics. However, a few languages do provide sets as primitive objects, the most famous being SETL, which

[2]

embraces standard mathematical notation more or less whole, and is the granddaddy of set-oriented languages.

Most computations performed by mathematicians and computer scientists are informed by set theoretical reasoning, if only implicitly. In order to automate such computations, a computer system must somehow embody the appropriate formal set theoretical and problem-specific knowledge, and have inferencing capability sufficient to put this knowledge to use. Bearing in mind the essential trade-off between expressiveness and decidability (not to mention *efficiency*), the formalism embodied would ideally be tailored to suit the problem at hand, postulating no more concepts, and providing no more inferencing resources than needed.

The book under review, according to its preface, is an "introductory-level text on computable set theory [that] provides a thorough, up-to-date, and comprehensive account of set-oriented symbolic manipulation methods suitable for automated reasoning," and is "addressed to graduate and postgraduate students, scholars, and researchers in computer science, artificial intelligence, logic, and computational mathematics." The main concern is with logical methods of reasoning about sets, as opposed to algorithms for their efficient construction and manipulation. Several formalisms, adapted to different application needs, are covered. Formalisms applicable to finite sets are stressed.

2 Summary of Contents

The book is divided into four parts comprising 14 chapters. This is a good place to mention that a detailed table of contents, links to related software and web sites, the beginnings of an errata list, and answers to a few exercises are available at the book's web site:

`http://turing.dipmat.unict.it/~cantone/SetTheoryForComputing`

2.1 Part I: Introduction

Chapter 1, What is Computable Set Theory? crisply describes the field in its earlier days as chiefly concerned with isolating "fragments of the set-theoretical language which are [algorithmically] decidable". As a non-expert, I was disappointed in seeking a similarly succinct answer to the title question for the subject as it stands today. Most of the first chapter is devoted to an overview of the rest of the book, however, which suffices as a kind of operational description of the field.

Chapter 2, Logical Background provides a condensed course in the basic logic needed in the sequel. Coverage is highly compressed with emphasis, of course, on computational aspects. Besides presenting the expected accounts of propositional and predicate calculi, the authors also treat "map" calculus, which is essentially the traditional calculus of (dyadic) relations. Map calculus is intermediate in expressiveness between propositional and predicate logic. Unlike either of the latter, it is also incomplete. Subsequent use of this formalism illustrates the tradeoffs in choosing a formalism; in this chapter, a theory of nested lists is expressed in both predicate and map calculi. Many results, some fundamental, are only briefly sketched, given as *obiter dicta*, or relegated to exercises, though more specialized computationally-oriented material such as reductions to normal forms and newer developments like Davis-Fechter quantifier elimination are considered at greater length.

2.2 Part II: Basics for Set-Theoretical Reasoning

Chapter 3, Axiomatic Views of Aggregates introduces axiomatic set theories beginning with the "kernel" first-order theory **K**, a rudimentary 3-axiom theory positing an empty set, a one-element insertion function, with, and an extensionality principle. Models of **K** must include some analog of the pure hereditarily finite sets, though infinite sets are not excluded. **K**'s lack of a foundation/regularity axiom means hypersets are not excluded, either. A map calculus rendition of **K** is also given. To arrive at a theory of pure finite sets, a series of axioms are added to the core theory **K**: ZF-like powerset, regularity and separation axioms, a Tarski-like finiteness axiom, and an axiom guaranteeing the existence of transitive closures, yielding altogether the theory $\mathbf{K}_{zer}$ ("*zer*" for Zermelo).

Alternatively, an $\in$-induction principle based on with is added to **K** to form the theory $\mathbf{K}_{\in-IND}$. The equivalence $\mathbf{K}_{zer} = \mathbf{K}_{\in-IND}$ is proved. Infinite sets are introduced via different versions of the infinity axiom (a terse discussion of ordinals, cardinals and cumulative hierarchies is conducted in side-panels so that these axioms may be better understood), a theory of finite hypersets is adduced, and ways to axiomatize sets with individuals (urelements) are shown. Examples are given of how to encode instances of common mathematical classes into set language, e.g., natural numbers (the usual von Neumann definition), ordered pairs (the usual Kuratowski definition), more general tuples, and graphs.

Chapter 4, Semantic Views of Aggregates considers models — in particular, computable models — of some axiomatic set theories of Chapter 3. A computable model is one in which every relation symbol is interpreted as a computable relation and every function symbol is interpreted as a computable function. The collection of pure hereditarily finite sets, $\mathcal{HF}$, is given as a computable model of $\mathbf{K}_{zer}$ and, perforce, **K**. Urelement-based models of $\mathbf{K}_{zer}$ are also considered. The basic model $\mathcal{HF}$ is extended to a computable model of **K** whose domain includes infinite sets. A model for the theory of finite hypersets is also constructed from $\mathcal{HF}$. Other topics included in this chapter: the treatment of sets in $\mathcal{HF}$ as numbers via Ackermann's numbering (= Rogers' canonical indices) of finite sets, bags (multisets), labeled trees and forests, and just enough coverage of a subset of the SETL language to follow some programming examples.

Chapter 5, Sets for Problem Solving shows how to use formal set theory in the specification and solution of practical problems, after introducing a series of notational conveniences for set-formation and finite binary relations and functions. A very neat example concerns the definition of languages accepted by deterministic and non-deterministic finite automata. Hypersets are also used to model deterministic finite automata. Additional examples cover the specification of mutilated checkerboard and Chinese checkers puzzles, and the SAT, topological sorting and single-source shortest path problems. SETL solutions of the three last-mentioned problems are provided. Among other topics covered in this chapter are the set-theoretic specification of numbers, tuples and trees as abstract data types, and a glimpse of program verification for algorithms given in set-theoretic terms.

Chapter 6, Decision Problems and Some Solutions covers fragments of various set-theoretic languages with respect to the satisfiability problem. The standard von Neumann universe **V**, the transfinite cumulative hierarchy based on $\emptyset$ that properly includes the hereditarily finite sets, is the preferred interpretive domain here. Assignments into **V** are styled "standard assignments", and a technique for obtaining a satisfying standard assignment from a satisfying assignment into

a cumulative hierarchy based on urelements is given. The satisfiability problem is solved for a class of formulas having restricted quantifiers: those formulas consisting of finite conjunctions with conjuncts of form $(\forall y_1 \in W_1) \cdots (\forall y_m \in W_m)\psi$ where $m \geq 0$, $y_1, \ldots, y_m$ are pairwise distinct and distinct from the W_i, and ψ is a propositional combination of atomic formulas of form $u = v$ or $u \in v$, with u and v both variables.

In contrast, it is shown that a Diophantine equation can be translated into a conjunction of set formulas, each having one of a few simple forms, such that the equation has a solution in integers iff the conjunction is satisfiable over sets; this reduction of Hilbert's Tenth Problem establishes the undecidability of the corresponding set-theoretic fragment. An efficient reduction of the NP-complete SAT problem to the set satisfiability problem is given, suggesting the hardness of solving the set satisfiability problem for various collections of set formulas. A *lot* more material is covered here: an introduction to unification, a version of the classical taxonomy of formulas based on quantifier prefixes for set-theoretic formulas with restricted quantifiers and its relation to the solvability of the satisfiability problem, etc. The chapter ends with an annotated list of about twenty decidable collections of set-theoretic formulas (such collections are called "syllogistics"). Some concepts mentioned in the annotations are undefined at this point in the text, though most are covered later in the text or in the references given.

Chapter 7, Inference Techniques and Methods lays the groundwork for the development of automated deduction systems that can exploit set-theoretic knowledge. Resolution methods are discussed, beginning with propositional resolution which is soon generalized to be relative to a theory, yielding "**T**-resolution". Conditions allowing **T**-resolution to be extended to first-order theories are then covered, and the proof of a generalized version of Herbrand's Theorem is sketched. Tableau-based theorem-proving methods for both propositional and predicate logic follow. Finally, an introduction to basic logic programming is given, including coverage of SLD-resolution and Prolog.

2.3 Part III: Decision Methods

Chapter 8, Set/Hyperset Unification Algorithms studies the extension of the unification method to set theoretical languages in ways that sensibly accommodate the extralogical symbols $\in$, $\emptyset$ and with. Initially, languages involving no other function symbols are treated; for these, a blind-search unification algorithm is given suitable for both sets and hypersets. An improved, goal-directed unification algorithm is then described, which allows terms including additional, uninterpreted ("free Herbrand") function symbols; however, this algorithm does not accommodate hypersets.

Chapter 9, A Syllogistic Solitaire explores a series of side-topics leading to a solution of the satisfiability problem for the language fragment consisting of unquantified set formulas over the monadic function symbol $\mathcal{P}$ (powerset), the dyadic function symbols $\cap$, $\setminus$ and $\cup$, and $\{\_, \ldots, \_\}$ (finite enumerations), all having the usual interpretations. The decision method is arrived at via an elaborate notion of games played on a "syllogism-board".

Chapter 10 Stratified Syllogistics presents decision methods for three "stratified syllogistics", set language fragments suitable for formalizing certain classes of assertions about stratified universes $U \cup \mathcal{P}(U) \cup \mathcal{P}(\mathcal{P}(U)) \cup \cdots$, where U is some non-empty set of individuals. Stratified languages include multiple distinct collections of variables, each intended to vary over a particular level in

some stratified universe; one collection varies over individuals, others over sets of individuals, etc. Stratification enables the fairly direct formalization of statements about common mathematical constructs. The satisfiability problem is solved for two different two-level syllogistics and one three-level syllogistic.

The simpler two-level syllogistic, **2LS**, is a quantifier-free language with two collections of variables, one for each of the two lowest strata of a stratified universe. Terms are constructed from set variables, constant symbols $\emptyset$ and **1** (to be interpreted as the empty set and the stratum of all individuals), and operators $\cap$, $\setminus$ and $\cup$; terms thus represent sets of individuals. Atomic formulas have form $x = y$, $x \in s$, $s = t$ or $s \subseteq t$, where x, y are individual variables and s, t are terms; formulas are simply propositional combinations of atomic formulas. An interpretation of **2LS** is the obvious thing. The syllogistic **2LST** adds the symbol $^-$ to **2LS**, whose intended interpretation is the topological closure operator of Kuratowski ($\overline{A}$ is the smallest closed set containing A). Armed with the new operator symbol, many other familiar topological operators, and some basic topological properties can now be expressed, e.g., the Int, Ext and ∂ (boundary) operators, and the properties of being open, closed, dense, etc. The satisfiability problem of **2LST** is reduced to that is **2LS**. The three-level syllogistic covered, **3LSSPU**, extends **2LS** by the addition of variables for collections of sets and the monadic singleton-, powerset- and unionset-forming operators ($\{\_\}$, $\mathcal{P}$ and $\bigcup$). As with several syllogistics treated in the book, its general satisfiability problem is reduced to that for conjunctions of literal formulas of a few simple types.

2.4 Part IV: Set-Specific Inference Engines

Chapter 11, Quantitative Set Reasoning shows how to translate set-satisfiability problems for the fragment "Multi-Level Syllogistic with a Cardinality operator" (**MLSC**) into satisfiability problems of the additive theory of arithmetic of natural numbers (Presburger arithmetic), a decidable theory. More specifically, the satisfiability problem for **MLSC** is translated into one or more problems solvable by methods of linear integer programming. **MLSC** is a fairly rich fragment of set theory, though one without quantifiers. The cardinality operator ($|\cdot|$) allows some arithmetic assertions to be formulated. The satisfiability problem of **MLSC** augmented with map-related symbols is shown to be reducible to that of **MLSC**. The additional language features are variables over maps (i.e., binary relations), and, for p a map variable and s any set term, the map-related terms $\mathsf{dom}\ p$, $\mathsf{im}\ p$, $p[s]$, $p^{-1}[s]$, $p\{s\}$ and $p^{-1}\{s\}$, ($p\{s\} = p[\{s\}]$ and $p^{-1}\{s\} = p^{-1}[\{s\}]$), and the atomic formulas $\mathsf{single\_valued}(p)$ and $\mathsf{one\_one}(p)$ — all with obvious intended interpretations. A reduction of the multi-level syllogistic satisfiability problem to that of propositional calculus is also given.

Chapter 12, Set Theory for Nonclassic Logics describes how to translate formulas of modal logic into set-theory. The key issue is the extension of the usual set-theoretical interpretation of the propositional connectives $\wedge$, $\vee$ and $\neg$ to the modal necessity operator $\Box$. A crash course in modal logic is provided to keep the book self-contained, though coverage is telegraphic. The main elements of the translation are the replacement of the accessibility relation (of the appropriate Kripke frame) by the membership relation $\in$ and the replacement of $\Box$ by the powerset operator $\mathcal{P}$. Under the assumption of frame-completeness, the translation is shown to be faithful, in that it preserves derivations. Minimal fragments of set theory that support the faithfulness of this translation are explored (a degree of minimality is necessary; the mapping of the accessibility relation to $\in$ means that the membership relation cannot in general be held to be acyclic or extensional — a target fragment cannot exclude hypersets). Extensions of the translation to other logics round out the

chapter.

Chapter 13, Logic Programming with Sets investigates adding support for reasoning about sets to the logic programming paradigm. In fact, an interpreter for the language discussed, {log} (read "set-log", suggesting Prolog + sets), has been implemented in Prolog, and is freely available from the book's web site, as are working sample applications and related papers offering more expansive coverage than the compressed treatment afforded here.

Chapter 14, Syllogistic Tableaux applies the tableau method to two fragments of set theory, presenting tableau-based calculi for **MLSS** ("Multi-Level Syllogistic with Singleton [set-formation]") and for its extension **MLSS**$\mathcal{F}^{\forall}$ ("**MLSS** with Function Symbols and Restricted Quantification"). Each tableau calculus is proved sound and complete w.r.t. the corresponding fragment, and a decision procedure for each fragment based on the related tableau calculus is supplied.

The language of **MLSS**, another quantifier-free theory, consists of a countable infinity of uninterpreted set constant symbols $\mathbf{c_0}, \mathbf{c_1}, \ldots$, a distinct set constant symbol $\emptyset$ (to be interpreted as the empty set), function symbols $\cup$, $\cap$, $\setminus$ and $\{\_\}$, and predicate symbols $\in$ and $=$ (all with the usual interpretations), and propositional connectives. A (set) term of **MLSS** is either a set constant symbol, or of form $t_1 \cup t_2$, $t_1 \cap t_2$, $t_1 \setminus t_2$, and $\{t_1\}$. The atomic formulas of **MLSS** are $t_1 \in t_2$ and $t_1 = t_2$, where the t_i are set terms. The fragment **MLSS** is the whole collection of **MLSS**-formulas (sentences, really), which are just the propositional combinations of atomic formulas.

The language of **MLSS**$\mathcal{F}^{\forall}$ extends that of **MLSS** with countable infinities of both set variables and uninterpreted monadic function symbols, and quantifiers. The definition of the fragment **MLSS**$\mathcal{F}^{\forall}$ is somewhat technical. **MLSS**$\mathcal{F}^{\forall}$ consists of those sentences φ of the expanded language in which each occurrence of a quantified subformula is positive (in the usual proof-theoretic sense) and has form $(\forall x \in t)\psi$, where

1. t is ground
2. the variable x doesn't occur as a proper subterm of any term t' in ψ that includes interpreted symbols or occurs on the L.H.S. of a membership literal.

The intended interpretation of both **MLSS** and **MLSS**$\mathcal{F}^{\forall}$ is over the von Neumann universe. **MLSS**$\mathcal{F}^{\forall}$ is fairly expressive; one can express a limited comprehension scheme, functions with multiple arguments (via Kuratowski's ordered pair definition), functional composition, and properties of functions such as injectivity and idempotency.

3 Opinion

The book contains a wealth of material, including both basic material and recent developments from the research literature, much of it the work of the authors or their collaborators. This supports the authors' intention that the book serve as both textbook and monograph. A useful variety of formalisms and examples are studied, with the treatment of map calculus as nice bonus. Many more topics than summarized above are touched on in the course of this text.

Coverage is quite dense, so as a textbook it's probably best suited for use in a special topics course, with a knowledgeable instructor and well-prepared students. The preface describes the book as "largely self-contained", and this is arguably true, but the onslaught of topics keeps motivation to a minimum. Perhaps more prior experience with both logic and formal set theory (and, in spots,

higher mathematics) than the authors suggest is required to appreciate the book, or solve some of the harder exercises.

Regarding exercises, there are nearly 200 of them. These are helpfully placed near coverage of the relevant material, rather than being collected at the ends of sections or chapters. Some are of the "complete the proof" variety that allows for more compact treatment of the related material. Exercises vary greatly in — but are not graded for — difficulty; however, those dependent on extra-textual knowledge are marked with asterisks (advanced sections and chapters are similarly marked). References accompany some of the more challenging problems.

Here are some other positive features:

- Engaging quotations appear under some chapter and section headings.
- Useful historical and bibliographical notes are gathered in the "Commentary" sections that conclude most chapters.
- A full bibliography with over 250 references.
- The book's web site, though as of this writing (May 2002) it's still a work in progress (for example, only 6 exercises have solutions posted, and the errata list is short).

On the negative side, a text this crowded with concepts demands razor-sharp prose throughout, but the narrative is thickish and opaque in places, even in the earlier chapters where clarity is so important for non-cognoscenti. While the authors' English is generally excellent, it is marred uncomfortably often by lapses in grammar or diction. (Some lapses are rather charming, e.g., "...wieldly chunks...", but not most.) The non-expert may suffer a bit of a "frame problem" — the context for various remarks is not always immediately clear.

The book's handling of term use versus term definition is occasionally infelicitous (e.g., the von Neumann universe is mentioned in several places well before being defined). Understanding some passages in non-asterisked sections requires knowledge of asterisked sections or unasterisked sections not yet covered. More than the usual percentage of misprints seem afoot, but this may be the effect of heightened vigilance provoked by the other shortcomings mentioned above.

Some other quibbles:

- I found the book's layout unappealing. Too many stretches mixing narrative with formulas, tables, panels or figures have a crowded look, making the text both visually and conceptually dense. At least one panel appears in the wrong section.
- The tables of symbols and abbreviations at the back of the book don't give page numbers where the terms are introduced.
- More definitions should be visually set off, rather than being submerged in the surrounding narrative; some definitions, significantly those of some syllogistics, could be more definitively defined.
- The effort to be complete sometimes results in clutter. In places, it is difficult to isolate the main line of thought from the side lines.
- Some bibliography entries cite out-of-print editions of texts still in print in other editions.

Quibbles notwithstanding, it is unlikely there is any single-volume substitute for **SET THEORY FOR COMPUTING**. Its publication is a service to the intended community of readers.

Review of: **Theory of Computational Complexity** [3]
512 pages, $105.00 dollars new, hardcover
by Authors: Ding-Zhu Du and Ker-I Ko
Publisher: John Wiley & Sons

Reviewed by E W Čenek, University of Waterloo

1 Overview

The study of computational complexity forms an active and central component of theoretical science, and has since its inception in the 60s. This study has expanded from the study of Turing-machine complexity theory to include various other computational models such as decision trees and circuits. These other models allow for a fine tuning of complexity classes and more accurate classifications of problems.

The field has also expanded from the study of deterministic algorithms that return a definite answer—if an answer is returned—to the study of algorithms that return answers that are probably true. Preferably, answers that are highly probable to be correct, for some value of highly.

2 Summary of Contents

The book is divided into three parts; uniform complexity, nonuniform complexity, and probabilistic complexity. The first part contains that material common to most books covering computational complexity. Part I contains four chapters: the first chapter introduces deterministic and non-deterministic Turing machines and defines the basic complexity classes, including P, NP, PSPACE and NPSPACE, LOGSPACE, and EXP. The Universal Turing Machine is introduced, as is the widely used diagonalization proof technique.

Next, Du-Ko discuss NP-completeness, introducing the class of NP and defining NP-completeness. Cook's theorem that SAT is NP-complete is proven, and reductions are given for the standard stable of NP-complete decision problems. They also discuss polynomial-time Turing reducibility to show that optimization problems such as Traveling Salesman are NP-complete.

Chapters 3 and 4 discuss the polynomial time hierarchy and the structure of the class NP, first covering those problems which are thought to be in $NP - P$ but are not NP-complete, and then introducing the concept of relativization, which discusses the complexity of a problem relative to a set of oracles. Interestingly, depending on the choice of oracles, it is possible to answer the question of $P = NP$ either way, relative to different oracles.

Part II discusses nonuniform computational complexity: what happens if the structure of computational model is simplified? The emphasis in this part is on proof technique, rather than result, introducing decision trees and circuits and showing how to prove lower bounds on complexity using them. Lastly polynomial-time isomorphism—a one-to and onto correspondence between instances of two problems which is both polynomially computable and polynomially invertible—is introduced in Chapter 7. Many natural NP-complete problems are polynomial-time isomorphic, leading to the conjecture that, within a polynomial factor, all NP-complete problems are effectively identical so that a heuristic for a single NP-complete problem will solve all NP-complete problems. Du-Ko discuss this conjecture and the related conjectures of EXP-complete and P-complete problems.

Part III considers the complexity of randomized computation. Using randomized algorithms to solve hard problems has become increasingly popular, leading to the need to analyze these algorithms appropriately. This leads to the introduction of probabilistic Turing Machines and probabilistic complexity classes, and the counting class #P which asks, for any problem, what the total number of different possible answers is. Thus #SAT asks what the total number of different Boolean assignments satisfying the formula is, rather than asking whether at least one such assignment exists.

In Chapter 10, Du-Ko take a somewhat different tack from counting, studying interactive proof systems. In an interactive proof system there are two agents—the prover and the verifier—and the goal of the prover is to convince the verifier that the answer x for a given problem A is correct. The prover and verifier alternate sending strings to one another, where a string may depend on previous strings and the original input. Since deciding what strings to send can amount to solving the intractable problem, the prover can demonstrate that x is probabilistically likely to be in A, by showing that if $x \notin A$ then there exist relatively many bad strings of length $q(|x|)$. Therefore the verifier consistently tries random strings of the appropriate length until either a counter example is found or the probability of a counter example existing and not being found is sufficiently small. The Arthur-Merlin Proof System is one such system where Arthur is a verifier with the power of a probabilistic Turing Machine, but the prover Merlin is so powerful that he can read the entire history of Arthur's computations, including the random numbers used. This leads to the AM complexity class, and some time is spent discussing the relative hierarchy of these new classes.

Interactive proof systems, in turn, lead to the notion of proof checking. Given a short proof, is it possible to check correctness in probabilistic polynomial time? This notion of probabilistically checkable proofs leads to a new characterization of NP, which is particularly applicable to the study of approximizing NP-hard combinatorial optimization problems.

3 Opinion

The "Theory of Computational Complexity" is written as a mathematical textbook, with all proofs included (albeit a few that are left as exercises), and should serve well as a graduate textbook. The first part covers those complexity topics that one expects to see in any computational complexity textbook. Parts II and III are reasonably independent from one another, as seems only reasonable given the scope of the two parts. The problems at the end of each chapter are interesting, and serve to extend the material.

I enjoyed reading the book; each section flowed into the next, and there was an overarching organization that made it easier to follow the material. There were a few minor typos that I detected, but those were obvious from context. Program-size, or Kolmogorov, complexity is not discussed in the book, which is a pity since Kolmogorov complexity would have produced an interesting unified view of nonunifom models, but is understandable considering the scope of the topic.

Computer Arithmetic Algorithms
by Israel Koren
Published by A.K. Peters
296 pages, $49.00

Reviewer: George A. Constantinides (george.constantinides@ieee.org) [4]

1 Introduction

This book covers the broad topic of implementing arithmetic in digital logic and/or in software.

With the increasing acceptance of custom hardware designs for arithmetic processors, due in no small part to the commercial success of the Field-Programmable Gate Array, a sound knowledge of computer arithmetic is becoming important for an ever-increasing circle of designers.

This review is probably somewhat unusual for SIGACT members as the theoretical aspects of arithmetic design, such as fundamental limits on the speed of computation, are covered only fleetingly by this book. This in no way detracts from the value of the book as an introduction to the field.

The review is of a second edition (2002) of a book originally published in 1993. Apart from general corrections and minor additions, new sections have been introduced on: floating-point adders, floating-point exceptions, general carry-look-ahead adders, prefix adders, Ling adders, and fused multiply-add units.

2 Book Contents

I have listed below those elements of each chapter which are particularly noteworthy.

2.1 Conventional Number Systems

Basic number systems and radix conversions

2.2 Unconventional Fixed-Radix Number Systems

Negative-Radix fixed number systems; Signed-digit number systems; Binary signed-digit numbers

2.3 Sequential Algorithms for Multiplication and Division

Sequential multiplication; Sequential division; Nonrestoring division; Square root extraction

2.4 Binary Floating-Point Numbers

Choice of floating-point representations; The IEEE floating-point standard; Roundoff schemes; Guard digits; Floating-point adders; Exceptions; Round-off errors and their accumulation

2.5 Fast Addition

Ripple-carry adders; Carry-look-ahead adders; Conditional sum adders; Optimality of algorithms and their implementations; Carry-look-ahead addition revisited; Prefix adders; Ling adders; Carry-select adders; Carry-skip adders; Hybrid adders; Carry-save adders; Pipelining of arithmetic operations

2.6 High-Speed Multiplication

Reducing the number of partial products; Implementing large multipliers using smaller ones; Accumulating partial products; Alternative techniques for partial product accumulation; Fused multiply-add unit; Array multipliers; Optimality of multiplier implementations

2.7 Fast Division

SRT division; High-radix division; Array dividers; Fast square root extraction

2.8 Division through Multiplication

Division by convergence; Division by reciprocation

2.9 Evaluation of Elementary Functions

The exponential function; The logarithm function; The trigonometric functions; The inverse trigonometric functions; The hyperbolic functions; Bounds on the approximation error; Speed-up techniques

2.10 Logarithmic Number Systems

Sign-logarithm number systems; Arithmetic operations; Comparison to binary floating-point numbers; Conversions to/from conventional representations

2.11 The Residue Number System

Arithmetic operations; The associated mixed-radix system; Conversion of numbers from/to the residue system; Selecting the moduli; Error detection and correction

3 Opinion

The target audience of this book should be advanced undergraduate or graduate students, for whom it would serve as an excellent introduction. Although introductory in nature, each chapter provides an extensive list of primary reference material, which should provide any graduate student with a head-start in the subject.

Highlights of the book include a very clear description of non-restoring division and its importance, and the treatment of floating-point representations. Floating-point is described with an emphasis on the design trade-offs involved in the selection of a standard such as the IEEE floating-point standard, rather than just presenting IEEE as a *fait-accompli*. Division is given the detailed treatment it deserves, occupying two out of the eleven chapters of the book. The inclusion of logarithmic number systems is also of particular interest.

What is really unusual in a book on computer arithmetic, is to include a treatment of how to approximate the elementary functions. This chapter is only partially developed, providing one detailed approach for each function considered, and would benefit from a detailed discussion of CORDIC, modern table-based approaches and hybrids. Having said this, any treatment of elementary functions is appreciated, and references are given to some of the standard text-books on the subject.

Overall, I would definitely recommend this book to anyone with a background encompassing basic digital design.

4 Additional Information

Curious readers may wish to look at the book web-site,
`http://www.ecs.umass.edu/ece/koren/arith`,
and a nice Java-based simulator covering many of the topics in the book can be found at
`http://www.ece.umass.edu/ece/koren/arith/simulator`.

Review [5] of
An Introduction to Quantum Computing Algorithms
Author: **Arthur O. Pittenger**
Hardcover: **152 pages**
Publisher: **Birkhäuser**, **2000** , **ISBN 0-8176-4127-0**
Progress in Computer Science and Applied Logic, V. 19

Reviewer: Andrea Marchini (`marchini@acm.org`)

1 Introduction

Quantum computing is a relatively recent field of computing sciences focusing on the application of quantum mechanical laws to computer technology. At the beginning of the 80s, the work of Benioff, Feynman and Deutsch built the foundations of this rapidly evolving area of study. The main advantages of quantum computing are the exponential computing power that quantum systems provide, due to the *superposition* of the states, and the possibility to communicate over large distances in a secure way, thanks to the *entanglement* of particles. Large number factorisation should be feasible with the Shor's algorithm and public key algorithms like RSA would be vulnerable, new physics simulation and other heavy computational problems could be solved. One of the most amazing aspect of quantum computing is the ability to obtain a result from a process dominated by the probability and the uncertainty. The *qubit*, the counterpart of the bit in a quantum system, can represent both a low and a high state simultaneously.

The building of a real quantum computing system meets different obstacles, from the collapse of the computation process caused by interferences, known as *decoherence*, to the reading of the computation result, that could destroy the result itself. In the last years, different quantum computers have been developed, many of them by means of NMR technologies. The last information

available is about a 7 qubit system successfully running the Shor's factoring algorithm. Many questions about quantum computing are still open, first of all the feasibility of an effective quantum computer, and the time needed to develop such a system.

Pittenger's book, as the title suggests, explains the mathematics at the basis of the quantum computing and the fundamental algorithms, including Shor's factoring, Grover's search and error correction algorithms. The reader won't find information about real quantum computers development nor will he be introduced to quantum mechanics.

2 Contents

Chapter 1, Quantum Statics

In the first chapter the author, after a quick survey of some important quantum mechanics evidences like the Stern-Gerlach spin measurement and the Einstein-Podolsky-Rosen paradox, introduces the static model of the system and the *Quantum Probability Space* with the spin 1/2 particles example. Then he explains the Dirac notation, the standard notation to work with high dimensional spaces. In the last paragraph he introduces unitary transformations, operations that can model the information without measuring it and therefore destroy it, and then shows the conformity of the model to the Heisenberg uncertainty principle.

Chapter 2, Basics of Quantum Computation

The second chapter, after the introduction of *tensor products* as the model to handle situations involving many qubits in one Quantum Probability Space, explains the *quantum gates*, the elementary brick of a quantum circuit. The author shows the XOR, the Toffoli gate and the way to combine gates to build a simple quantum circuit, an adder. The last paragraph quickly presents the teleportation circuit, an application of the entanglement in the quantum communication field.

Chapter 3, Quantum Algorithms

The following fundamental quantum algorithms are described, with highlights on their ability to solve problems more efficiently than classical algorithms:

- Deutsch-Jozsa: having two classes of functions that map V^n to {0,1}, where V^n is the linear space of n long binary vector, one class mapping to a constant value, the second mapping 2^{n-1} vectors to 0 and 2^{n-1} to 1, the algorithm distinguishes between the two classes calling the function twice.

- Simon: the algorithm finds the value of an unknown n long binary vector ξ using a function f, $f(x) = f(y)$ if and only if $x = y \bigoplus \xi$, where $\bigoplus$ is mod(2) addition. The algorithm uses the periodic properties of the function to enhance the probability to evaluate ξ.

- Grover: having a "needle in the haystack" function f that map to 1 only one vector out of 2^n, to 0 the remaining $2^n - 1$ vectors of the domain space V^n, the algorithm finds the "needle" with $O(\sqrt{N})$ evaluations, where $N = 2^n$, instead of $O(N)$ required by classical algorithms.

- Shor: the algorithm factors a number N with a work factor that is polynomial in $\log N$. Shor's algorithm raised the interest in quantum computing since some public key algorithms, like RSA, are based upon the difficulty of factoring large numbers. The author first applies the algorithm to the $N = 15$ case, then to the general $N = pq$ case.

- Finite Fourier Transform: the Fourier transform plays a key role in quantum algorithms; periodicity is used to evaluate the probability of getting the correct result. The work factor of this quantum algorithm is polynomial in n, the number of the affected qubits.

The author then analyzes the structure of some of the above mentioned algorithms and explains the use of the eigenvectors of unitary transformations to perform the computations. The Bernstein-Vazirani version of the Deutsch-Jozsa oracle, that needs just one call to the evaluated function, and the generalized version of the Simon's algorithm are examples of this different approach.

In the last paragraph the author, using again the Simon's and Shor's algorithms, shows that many quantum algorithms can be considered as special cases of a more general group theory.

Chapter 4, Quantum Error-Correcting Codes

The *Schrödinger equation* describes the dynamic evolution of the system when it comes to the real case. As a result of the interactions with the environment, the loss of superposition of the states, called *dechoerence*, intervenes and the computing system collapses. Quantum error correcting algorithms are therefore necessary in a real quantum computing system and the author presents the Shor's nine qubit error-correcting algorithm, that entangles ancillary qubits in order to achieve a 3 bit repetition scheme, and the Calderbank-Shor-Stean seven qubit error-correcting algorithm, based upon the classical Hamming code. The five qubit algorithm developed by Laflamme, follows as an example of an encoding scheme not based on a classical code. These algorithms are special cases of the theory of stabilizers groups, presented in the following paragraph. In the last section the author explains how to develop quantum error correcting codes combining different classical encoding schemes.

3 Opinion

Since quantum computing is a highly interdisciplinary science, the author has tried to capture the attention of a large variety of readers and he has mostly achieved this objective. The book can be used as a formal introductory text for graduate students as well as a fascinating, but still engaging resource for interested readers who are comfortable with linear algebra. A basic knowledge of quantum mechanics is necessary to appreciate the content of the book, and a firm comprehension of the main concepts of physics is desirable. Pittenger helps the reader into focusing the attention on the algorithimic aspects rather than the formal content and uses examples as integral part of the book, illustrating the substantial meaning of the quantum theory applied to computing. He also proposes some exercises to stimulate an insightful reading. Some of them just help to recall the basilar knowledge, while others are more challenging and prepare the understanding of the most complex aspects of the theory.

The bibliography is complete and the interested reader can improve the understanding of the book and of the entire matter by following the numerous references, acquiring in this way more tools for the comprehension of a subject of such complexity. In the last two chapters the author approaches the theory from the particular to the general case, and this is appreciable, but the feeling is that he could have spent some more pages about the group theory and the generation of quantum error correcting code to give the less expert reader a more clear view of the abstract theory behind quantum computing.

Nevertheless the book seems really to be the "useful self-study guide" the author promised to write and it is a serious way to give a first look at this wonderful matter.