

- 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 NP and defining NP-completeness. Cook's theorem that SAT is NP-complete is proven, and reductions are given for the standard stable

³© E. W. Čenek, 2003.

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 approximating 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 nonuniform 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.

⁴©2003 George A. Constantinides