Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science' (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

Also in this series

Iain D. Craig
Object-Oriented Programming Languages: Interpretation
978-1-84628-773-2

Max Bramer
Principles of Data Mining
978-1-84628-765-7

Hanne Riis Nielson and Flemming Nielson
Semantics with Applications: An Appetizer
978-1-84628-691-9

Michael Kifer and Scott A. Smolka
Introduction to Operating System Design and Implementation: The OSP 2 Approach
978-1-84628-842-5

Phil Brooke and Richard Paige
Practical Distributed Processing
978-1-84628-840-1

Frank Klawonn
Computer Graphics with Java
978-1-84628-847-0

David Salomon
A Concise Introduction to Data Compression
978-1-84800-071-1

David Makinson
Sets, Logic and Maths for Computing
978-1-84628-844-9

Alan P. Parkes

A Concise Introduction to Languages and Machines

Springer

Alan P. Parkes, PhD
Honorary Senior Lecturer, Computing Dept., Lancaster University, UK

Undergraduate Topics in Computer Science ISSN 1863-7310
ISBN 978-1-84800-120-6 e-ISBN 978-1-84800-121-3
DOI 10.1007/978-1-84800-121-3

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 0000000

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

Springer Science+Business Media
springer.com

Contents

Preface

Aims and Objectives

This book focuses on key theoretical topics of computing, in particular formal languages and abstract machines. It is intended primarily to support the theoretical modules on a computer science or computing-related undergraduate degree scheme.

Though the book is primarily theoretical in nature, it attempts to avoid the overly mathematical approach of many books on the subject and for the most part focuses on encouraging the reader to gain an intuitive understanding. Proofs are often only sketched and, in many cases, supported by diagrams. Wherever possible, the book links the theory to practical considerations, in particular the implications for programming, computation and problem solving.

Organisation and Features of the Book

There is a short introductory chapter that provides an overview of the book and its main features. The remainder of the book is in two parts, *Languages and Machines* and *Machines and Computation*.

Part 1, Languages and Machines, is concerned with formal language theory as it applies to Computer Science. It begins with an introduction to the notation and concepts that support the theory, such as strings, the various ways in which a formal language can be defined, and the Chomsky hierarchy of formal language. It then focuses on the languages of the Chomsky hierarchy, in each case also introducing the abstract machines associated with each type of language.

The topics are regular languages and finite state recognisers, context free languages and pushdown recognisers, and context sensitive and unrestricted languages and the Turing machine. Many important theoretical properties of regular and context free languages are established. The more intellectually demanding of these results are mostly confined to a single chapter, so that the reader can focus on the general thrust of the argument of Part 1, which is to demonstrate that the Chomsky hierarchy is indeed a proper hierarchy for both languages and machines, and to consider some of the implications of this.

In the first part of the book the finite state machine, the pushdown machine and the Turing machine are considered as language recognisers, though many hints are given about the potential computational properties of these abstract machines.

Part 2, Machines and Computation, considers the computational properties of the machines from Part 1 in more detail. The relationship between finite state machines and the digital computer is explored. This leads us on to the need for a more powerful machine to deal with arbitrary computation. This machine is shown to be the Turing machine, introduced in Part 1 as a language processor.

The Turing machine is used to explore key aspects of computation, such as non-determinism, parallel processing and the efficiency of computational processes. The latter is considered in the context of a brief introduction to algorithm analysis, using big O notation.

The book also considers the limitations of computation, both in terms of language processing (simply defined formal languages that cannot be processed by any machine) and computation (the halting problem and related issues).

The book contains numerous illustrative figures, and proofs are often partly accomplished through diagrammatic means.

From Chapter 2 onwards, each chapter concludes with exercises, some of which are programming exercises. Solutions and hints for many of the exercises appear at the end of the book.

The book also contains a list of recommended reading material.

For Students

I wrote this book partly because when I studied this material as part of my own Computing degree, I had to work really hard to understand the material, a situation which arose not because the material is too difficult, but because it was not well presented and the books seemed to assume I was a pure mathematician, which I am not.

This book is primarily for undergraduate students of computing, though it can also be used by students of computational linguistics and researchers, particularly those entering computer science from other disciplines, who find that they require a foundation or a refresher course in the theoretical aspects of computing.

Some aspects of the book are certainly clearer if the student has some experience of programming, though such experience is not essential for understanding most of the book.

The reader is advised where especially demanding material can be omitted, though he or she is encouraged to appreciate the implications of that material, as such an appreciation may be assumed later in the book.

For Instructors

I wrote this book partly because when I taught this material to undergraduates, as a Computer Science lecturer in a UK university, I had to work really hard to present the material in an appropriate way, and at the appropriate level, a situation which arose mainly because the available books seemed to assume the students were pure mathematicians, which mostly they are not.

This is intended to be not only a book to promote understanding of the topics for students, but also a recommendation of the core material that should be covered in a theoretical computer science module for undergraduates. I suggest that to cover all of the material would require at least a module in each semester of one year of the degree course. If possible, the linguistic aspects should precede a compiler course and the computational aspects should precede an algorithm analysis course. If the student has programming experience the material is much more accessible. This should influence the way it is presented if it appears early on in the degree scheme.

There is, of course, much more theory relating to programming language design, compiler writing, parallel processing and advanced algorithm analysis than presented in this book. However, such additional theory is best covered in the modules that concern these subjects, some of which are often optional in a computing degree scheme.