COMPUTABILITY, COMPLEXITY, LOGIC

E. BÖRGER

COMPUTABILITY, COMPLEXITY, LOGIC

E. BÖRGER
Department of Informatics
University of Pisa
Pisa, Italy

NORTH-HOLLAND
AMSTERDAM • NEW YORK • OXFORD • TOKYO

ELSEVIER SCIENCE PUBLISHERS B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

Distributors for the U.S.A. and Canada:

ELSEVIER SCIENCE PUBLISHING COMPANY, INC.
655 Avenue of the Americas
New York, N.Y. 10010, U.S.A.

Library of Congress Cataloging-in-Publication Data

Börger, E. (Egon), 1946-
[Berechenbarkeit, Komplexität, Logik. English]
Computability, complexity, logic / E. Börger.
p. cm. -- (Studies in logic and the foundations of mathematics ; v. 128)
Translation of: Berechenbarkeit, Komplexität, Logik.
Bibliography: p.
Includes index.
ISBN 0-444-87406-2
1. Computable functions. 2. Computational complexity. 3. Logic, Symbolic and mathematical. I. Title. II. Series.
QA9.59.B6713 1989
511.3--dc20 89-33636
CIP

ISBN: 0 444 87406 2

This volume is a translation of 'Berechenbarkeit, Komplexität, Logik' which was published by F. Vieweg & Sohn Verlagsgesellschaft GmbH. It has been, translated into the English language and prepared for offset printing by J. C. Harvey.

PRINTED IN THE NETHERLANDS

This book is dedicated to

Donatella Barnocchi

and

Dieter Rödding

(*24. 8. 1937, †4. 6. 1984)

To both of them I owe more than this book - its beginning, its being completed and the best of its contents. I owe them, in particular, their example: it consists in confronting persons and situations in life and science selflessly and with an open mind, and never abandoning the purpose of recognising what is essential and true and to think and act accordingly.

PREFACE

The theme of this book is a pair of concepts, already recognised as belonging together by Leibniz, whose mathematical development from Frege to Turing has laid the theoretical foundation of computer science: the concept of formal language as carrier of the precise expression of meaning, facts , problems, and the concept of algorithm or calculus, that is, formally operating procedure for the solution of precisely described questions and problems. The book gives a unified introduction to the modern theory of these concepts, to the way in which they developed first in mathematical logic and computability theory and later in automata theory, the theory of formal languages and complexity theory. Apart from considering the fundamental themes, and nowadays classical aspects of these areas, the subject matter has been selected to give priority throughout to the new aspects of traditional questions, results and methods which have developed from the needs or knowledge of computer science and particularly of complexity theory.

The aim of this book is twofold: to be a *textbook* for introductory courses in the above-mentioned disciplines as they occur in almost all current curricula of computer science, logic and mathematics, but apart from this, to be a **monograph** in which further results of new research (to a large extent in textbook form for the first time) are systematically presented and where the attempt is made to make explicit the connections and analogies between a variety of concepts and constructions. A price must be paid by the reader for the knowledge I expect him to acquire when and if the experiment is successful: for the beginner the first lectures of the text will be difficult due to the profusion of concepts, remarks and forward and backward references to currently posed clusters of problems - particularly if he approaches the material by self-study unaccompanied by lectures. My advice is to initially skip over those parts which, despite study, are not understood; the connections will spring to mind on second reading.

The following **remarks on the use of the book** might be helpful; I have employed all parts of this book as the basis of introductory or advanced lectures on the foundations of theoretical computer science, automata theory and formal language, logic, computability- and complexity-theory. To enable the reader to recognise the use and interdependence of the various parts I have devised a detailed table of contents and a graph of interdependence. The sections marked with *

contain material which is not treated in the basic courses but is suitable to follow them.

The ***arrangement of propositions*** as theorem, lemma, remark and exercise mirrors the methodical significance of the various states of affairs from a contemporary point of view. It says nothing about historical or individual achievements to have proved these propositions for the first time. Many a significant proposition becomes a simple example as a result of later progress.

I strongly recommend beginners to work out with pencil and paper, at first reading, all matters of routine or intermediate steps which are not explained in detail and to solve the exercises, or at least, try to solve them. By doing this one not only learns whether one has really understood the preceding subject matter and how to apply it, but one also acquires a feeling for what is essential in the techniques used. In this endeavour it might help that I have tried to express complicated ideas occurring in proofs without the use of formulas. The reader is advised to use this method of intuitive, but precise substantive thinking which opens the way to a deeper understanding.

The ***references to literature*** at the end of each section are considered as completions of those references given in the text.

I would like to express my heart-felt thanks to the many persons who have helped with the work on this book in the past years, by no means all of whom I am able to mention. I name in particular the following colleagues and collaborators who read the manuscript in whole or in part and who have given me valuable criticisms: K. Ambos-Spies, H. Brähmik, T. Brand, A. Brüggemann, H. Fleischhack, J. Flum, G. Hensel, H. Kleine-Büning, U. Löwen, L. Mancini, K. May, W. Rödding, H. Schwichtenberg, D. Spreen, J. Stoltefuß, R. Verbeek, S. Wainer.

Separately I would like to thank: K. Ambos-Spies, whose elaboration of one of my Dortmund logic lectures I have partially used in chapters D/E, and who has given valuable help, particularly to §BII3; U. Löwen for a critical reading of the entire manuscript and the preparation of the symbol- and subject-indices; K. May for careful corrections and numerous drawings; H. W. Rödding for the intricate control of the bibliography.

I would especially like to thank U. Minning, R. Kühn, J. Kossmann, P. Schoppe and K. Gruhlich for the precise transposition of parts of several versions of my manuscript into the type-script for the printer. U. Minning has borne the main burden in this - her engaging and friendly manner has often allowed me to forget this arduous labour.

Finally, but not less heartily, I thank Walburga Rödding and many other colleagues who, in the past difficult six

weeks, have given me their spontaneous moral support, thereby decisively helping me to complete this book.

Dortmund, 3. 7. 1985 EGON BÖRGER.

Note on the second edition. At this point I would like to express heartfelt thanks to M. Kummer, P. Päppinghaus, V. Sperschneider: mainly because of their list of errors corrections have been made in the second edition. At this point also I thank in advance all those readers who show me further errors.

Pisa, Spring 1986 EGON BÖRGER.

INTRODUCTION

To the towering achievement of the mathematics of the last one hundred years belongs the formulation of a precise, embracing concept of formal language and a general concept of algorithm.

Already ***Leibniz*** had recognised that the creation of a mathematically precise universal language for the expression of arbitrary statements (***characteristica universalis***) was related to the development of a sufficiently general (concept of) calculus (calculus ratiocinator) with regard to purely computable, formal - we would nowadays say algorithmic - decisions in scientific problems. To this corresponds the distinction, frequently made in linguistics, between ***descriptive*** and ***imperative*** elements or use of a language. Language or elements of a language can, on the one hand, be used for the description of states of affairs or facts, and on the other hand for the formulation and communication of directions (instructions) for the transformation (computation) of states of affairs and the construction (generation) of new states; such transformations include the solution of problems by the testing of objects for given properties (the so-called decision procedures).

Mathematics yields a classical ***example*** of this distinction with ***two basic types of mathematical problem formulation***. One type of problem is to formulate statements about objects inside a mathematical language and prove them in a mathematical system (later formalised). The other is to specify instructions for computation or generation (enumeration) of objects within an algorithmic language. Thus, it can be proved that any two natural numbers have a greatest common divisor, or a procedure for generating the greatest common divisor can be developed. This example shows how close the connection between the descriptive and imperative elements of a language can be: to prove a mathematical sentence α from an axiom system Ax by means of the given rules R of a formal system, means to determine the truth-value of the sentence "α follows from Ax by means of R" by specifying how α is finally obtained from Ax by the formal transformations allowed in R. The proof of the sentence "To each x there exists a y with the property E" can consist of a specification of a procedure which, for arbitrary x, produces a y with the property E (that is, of the description of such a process including proof of its efficacy.)

Another representative ***example*** - a typical phenomenon for the development of algorithmic problem-solving methods - is

the following: frequently, the main difficulty in the solution of a problem by computer program consists of circumscribing, bounding this problem exactly, excluding ewhat is not intended to be part of the problem; tyhis is what is commonly called "specifying" a problem. The efficiency of a programming language and the degree of correctness of the programs (describing algorithmic processes)formulated in it then depends essentially on the quality of the specification language as a vehicle of description and on the reliability of the methods by which programs are constructed from the specifications. The development of programming languages in the past thirty years shows very clearly that the descriptive and imperative use of language elements have an intrinsic connection; the kernel of the demands of the ever advancing programming language PROLOG rests on a single, flexible language - first order logic - being simultaneously the specification- and the programming-language, on one and the same object able to be a statement (description of a problem in the domain of a logic-language) and a program (algorithm for the solution of this problem in the domain of a programming language.)

The present, already extensive mathematical theory of the concepts of algorithm and formal (logic-) language has decisively influenced, conceptually and methodically, the development of the way in which one deals with programmable computing equipment, and this influence promises to increase rather than weaken in the future. From the conviction that a mathematical theory loses nothing in intellectual interest by helping one understand a part of reality, I have undertaken in this book to give an introduction to algorithm theory and logic, oriented to the requirements of computer science without abandoning valuable traditions of the history of thought or sacrificing mathematical merit.

The ***structure of the book*** is therefore as follows: the first book is devoted to the theory of algorithms, the second to logic. The ***theory of algorithms*** (also known as computability theory) in its modern form is, above all, the theory of the extent and complexity of classes of algorithms and the automata and machines that realise them. It answers such questions as : What is the meaning of "algorithm", "universal programming language", "programmable computing equipment"? (Ch. A) What are the principal, general limitations of algorithmic problem-solving methods and what role is played in this area logical or algorithmic means of description? (Ch. B). How can the efficiency and variety of algorithms be ordered hierarchically according to criteria which characterise algorithms by the available resources or purely syntactically by their structure ? (Ch. C)

The basic questions of the book on logic are correspondingly: Can mathematical precision be given to the idea of an assertion being true independently of its eventual meaning and only on the basis of its logical structure, and

can such a logical concept of truth be characterised algorithmically ? (Ch. D) Is there a general algorithmic form of mathematical deduction from given premises ?(Ch. E) How are the universality of a logical language and the universality of a programming language related ? What is the relation between the expressibility of a logical language and the range of algorithms represented in it ? What is the connection between the syntactical logical complexity of expressions and the computation complexity of algorithms represented by them ? (Ch. F.)

TERMINOLOGY AND PREREQUISITES

We presuppose that the reader has such **mathematical maturity** as could be acquired by a one- or two-semester introductory course, although we seldom assume any specific knowledge apart from the principle of inductive definition and proof, and elementary facts of set theory. Where such special items occur, the mathematical facts used are explicitly mentioned and the reader can find them in standard introductory mathematical texts.

We therefore use the usual **set-theoretical notation** and symbolism:

$\subseteq$, $\supset$ are *inclusion* and *strict* or *proper inclusion* relations between sets. Thus $X \supseteq Y$ and $Y \subseteq X$ both mean that the set X includes Y. $X \supset Y$ and $Y \subset X$ both mean that X properly includes Y, ie $X \supseteq Y$ and $X \neq Y$.

$C(X)$ is the complement of the set X.

The **logical operations** will be written as follows:

for *negation*:	not
for *disjunction*:	or
for *implication*:	implies
for *conjunction*:	&, and
for *equivalence*:	iff, exactly when

For the *definitions* we write $x := y$ to mean that x is *defined* to be equal to y.

$\mathcal{N}$ denotes the set of natural numbers. The usual notation for **arithmetical operations** on $\mathcal{N}$ is assumed except that for more complicated expressions x or y a more readable expression is obtained by writing exponentiation as $x\#y$ rather than as the usual x^y (x to power y). The symbol $\#$ is occasionally used for different purposes, but its meaning should be clear from the context.

When not otherwise stated, by a **function** we shall always mean a partial function $f \subseteq A \times B$ which does not necessarily have a defined value $f(x)$, that is a $y \in B$ such that $(x, y) \in f$, for each $x \in A$. This set-theoretical conception of functions means the frequently useful identification of f with its *graph*, $G[f] = \{(x, y);\ y = f(x)\}$. We write:

$f(x)\downarrow$ iff $(\exists y \in B): (x, y) \in f$ (read : $f(x)$ is defined)
$f(x)\uparrow$ iff *not* $f(x)\downarrow$ (read : $f(x)$ is undefined)

A *function* $f \subseteq A \times B$ is called *total* iff $(\forall x \in A): f(x)\downarrow$. We use the *identity-symbol for partial functions* f, g based on the equality-sign for defined values in the sense:

$$f(x) = g(x) \quad \text{iff}$$

$$(f(x)\downarrow \text{ iff } g(x)\downarrow) \ \&\ (f(x)\downarrow \text{ implies } f(x)=g(x))$$

$$f = g \quad \text{iff} \quad (\forall x): f(x) = g(x)$$

By the notation $f: A \to B$ we mean that f is a function with domain of definition included in A and whose range of values is included in B.

Frequently we shall use the so-called **λ-notation**: for an arbitrary term t in which, besides x, other variables may occur in the role of parameters, $\lambda x.\, t(x)$ denotes that function which assigns the value $t(x)$ to x. Correspondingly, we write $\lambda x.\, P(x)$ for predicates (relations or properties). For the *parametrisation* of a function by "parameters" (parts of its sequence of arguments) $\mathbf{x}$ we write $f_{\mathbf{x}} := \lambda y.\, f(\mathbf{x}, y)$.

The place-number n of functions or predicates we shall give in the forms $f^{(n)}$, $P^{(n)}$ respectively, where appropriate. Where such a notation is lacking it is assumed that a proper determination of the number of argument places has been defined. Also, for a class F of functions we write $F^{(n)}$ for $\{f^{(n)};\ f \in F\}$.

For the *characteristic function* of a predicate P we write ξ_P, where $\xi_P(x) \in \{0, 1\}$ and $\xi_P(x) = 1$ in case $P(x)$ (read: P holds on x), $\xi_P(x) = 0$ otherwise.

We shall frequently use the following two operations of *iteration of functions* f and g:

$$f^0 := id \text{ (identity function)}, \quad f^{n+1} := f \circ f^n$$
$$Iter(f)(x, n) := f^n(x)$$

By the *iteration of* f *by* g, written $(f)_g$, we mean "iteration of f until g takes the value 0 on the computed value", that is

1. $(f)_g(x)\downarrow$ iff $(\exists n \in \mathbb{N}): f^n(x)\downarrow \ \&\ g(f^n(x))=0$.
2. If $(f)_g(x)\downarrow$, then $(f)_g(x) = f^n(x)$ holds for the smallest n with $g(f^n(x)) = 0$.

By an **alphabet** A we understand a finite, non-empty set $\{a_1, \ldots, a_m\}$ whose elements are called *symbols* or *letters*. A finite sequence of letters (from A) we call a *word* (on A) and write $A^* := \{w;\ w \text{ word on } A\}$. The *empty word* (that is, the sequence of length 0 of symbols) we denote by Λ. We put $A^+ := A^* - \{\Lambda\}$. The length of the word w is denoted by $|w|$

CONTENTS