

# 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

© ELSEVIER SCIENCE PUBLISHERS B.V., 1989

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.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher, Elsevier Science Publishers B.V./Physical Science and Engineering Division, P.O. Box 103, 1000 AC Amsterdam, The Netherlands.

Special regulations for readers in the U.S.A. - This publication has been registered with the Copyright Clearance Center Inc. (CCC), Salem, Massachusetts. Information can be obtained from the CCC about conditions under which photocopies of parts of this publication may be made in the U.S.A. All other copyright questions, including photocopying outside of the U.S.A., should be referred to the publisher.

No responsibility is assumed by the publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein.

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. Brähmík, T. Brand, A. Brüggemann, H. Fleischhack, J. Flum, G. Heneel, 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 SBII3; 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. Grulich 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. Pöppinghaus, 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  $\alpha$  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 " $\alpha$  follows from  $Ax$  by means of  $R$ " by specifying how  $\alpha$  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 what is not intended to be part of the problem; this 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:

$\supseteq$ ,  $\supset$  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  $Y$ , i.e.  $X \supseteq Y$  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  $y$ .

$N$  denotes the set of natural numbers. The usual notation for arithmetical operations on  $N$  is assumed except that for more complicated expressions  $x$  or  $y$  a more readable expression is obtained by writing exponentiation as  $x \uparrow y$  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,  $Gf = \{(x, y); y = f(x)\}$ . 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:

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

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

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

By the notation  $f: A \rightarrow B$  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  $\lambda$ -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)  $x$  we write  $f_x := \lambda y. f(x, y)$ .

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  $\xi_P$ , 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  $g$ :

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

$$\text{Iter}(f)(x, n) := f^n(x)$$

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  $\{a_1, \dots, a_n\}$  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  $\Lambda$ . We put  $A^+ := A^* - \{\Lambda\}$ . The length of the word  $w$  is denoted by  $|w|$ .

# CONTENTS

Introduction	XV
Terminology and prerequisites	XVIII

## Book One

<b>ELEMENTARY THEORY OF COMPUTATION</b>	<b>1</b>
Chapter A <b>THE MATHEMATICAL CONCEPT OF ALGORITHM</b>	<b>2</b>
<b>PART I.    CHURCH'S THESIS</b>	<b>2</b>
81. <i>Explication of Concepts.</i> Transition systems, Computation systems, Machines (Syntax and Semantics of Programs), Turing machines, structured (Turing- and register-machine) programs (TO, RO).	2
82. <i>Equivalence theorem.</i> $F_{\mu} \subseteq F(TO) \subseteq F(TM)$ , LOOP-Program Synthesis for primitive recursive functions, $F(TM) \subseteq F(RO) \subseteq F_{\mu}$ .	26
83. <i>Excursus into the semantics of programs.</i> Equivalence of operational and denotational semantics for RM-while programs, fixed-point meaning of programs, proof of the fixed-point theorem.	34
84*. <i>Extended equivalence theorem.</i> Simulation of other explication concepts: modular machines, 2-register machines, Thue systems, Markov algorithms, ordered vector addition systems (Petri nets), Post calculi (canonical and regular), Wang's non-erasing half-tape machines, word register machines.	37
85. <i>Church's Thesis</i>	48
3	
<b>PART II.    UNIVERSAL PROGRAMS AND THE RECURSION THEOREM</b>	<b>51</b>
81. <i>Universal programs.</i> Kleene normal form. Acceptable universal programming system and effective program transformations.	51
82. <i>Diagonalisation method.</i> Recursion theorem: fixed-point meaning (theorem of Rice), recursion meaning (implicit definitions: recursive enumeration of $F_{prim}$ , injective translation functions in Gödel numbering, isomorphism theorem for Gödel numberings, self-reproducing programs), parametric effective version with infinitely many fixed points.	58

Chapter B	COMPLEXITY OF ALGORITHMIC UNSOLVABILITY	68
PART I.	RECURSIVELY UNSOLVABLE PROBLEMS (Reduction method)	68
§1.	<i>Halting problem <math>K</math></i> Special cases of Rice's theorem.	69
§2.	<i>Simple reductions of <math>K</math></i> Decision problems of universal computing systems, Post's correspondence problem, Domino problem, Rødding's path problem.	71
§3*.	<i>Exponential diophantine equations.</i> Simulation of RO.	82
§4*.	$\lambda x, y, z. x = y^z$ is diophantine. Pell equations.	89
PART II.	THE ARITHMETICAL HIERARCHY AND DEGREES OF UNSOLVABILITY	103
§1.	<i>Recursively enumerable predicates.</i> Representation theorem. Universality.	103
§2.	<i>Arithmetical hierarchy.</i> Enumeration- and hierarchy-theorems, representation theorem, determination of complexity (infinity and cardinality statements, arithmetical truth-concept)	108
§3*.	<i>Reduction concepts and degrees of unsolvability.</i> Reduction concepts (theorem of Post), index sets (theorem of Rice and Shapiro, $\Sigma_n$ -complete program properties), creativity and $\Sigma_1$ -completeness (theorem of Myhill), simple sets ( $\Sigma_1$ versus $\Sigma_m$ versus $\Sigma_{n,n}$ , theorem of Dekker and Yates), priority method (theorem of Friedberg and Mučnik), complexity of the arithmetical truth concept.	114
PART III.	ABSTRACT COMPLEXITY OF COMPUTATION	144
§1.	<i>Speed-up phenomena.</i> Abstract measures of complexity, Blum's speed-up theorem, impossibility of effective speed-up.	145
§2.	<i>Functions of arbitrary complexity.</i> Theorem of Rabin-Blum-Meyer on functions of arbitrarily large program- or computing-time complexity, Blum's program-shortening theorem, gap theorem, union theorem.	155
§3*.	<i>Decomposition theory for universal automata.</i> Characterisation of the run-time-, input-, output-transition-, and stop-functions of universal automata; impossibility of uniform recursive simulation bounds on universal automata.	162

Chapter C	RECURSIVENESS AND COMPLEXITY	172
PART I.	COMPLEXITY CLASSES OF RECURSIVE FUNCTIONS	173
80.	<i>The k-tape Turing-machine model.</i> Tape reduction, tape- and time-compression, simulation complexity of a universal program.	173
81.	<i>Time- and place- hierarchy theorems.</i> Theorem of Fürer.	182
82.	<i>Complexity of non-deterministic programs.</i> Theorem of Savitch.	191
PART II.	COMPLEXITY CLASSES OF PRIMITIVE RECURSIVE FUNCTIONS.	196
81.	<i>Grzegorzczk hierarchy theorem</i> Equivalence of the characterisation by growth-rate (limited recursion, excursus on Ackermann branches), recursion- and loop-depth, computing-time complexity from Kleene normal form with polynomially bounded or $R_n$ -coding functions.	197
82*.	<i><math>E_n</math>-Basis- and <math>E_n</math>-computing time hierarchy theorem</i>	211
83*.	<i>Ackermann function and Goodstein sequences.</i> Theorem of Goodstein, Kirby and Paris.	220
PART III	POLYNOMIALLY- AND EXPONENTIALLY- BOUNDED COMPLEXITY CLASSES.	224
81.	<i>NP-complete problems.</i> Halting-, domino-, partition-, knapsack-, clique-, Hamiltonian cycle-, travelling salesman-, integer-programming-problems.	226
82.	<i>Complete problems for PTAPE and exponential classes.</i>	240
PART IV	FINITE AUTOMATA	242
81.	Characterisation by (non-)deterministic <i>acceptors and regular expressions.</i> Theorems of Rabin and Scott, Kleene.	242
82.	Characterisation by <i>indistinguishability congruence relation</i> Theorem of Myhill and Nerode with corollaries (state minimisation, examples of non-regular languages, loop lemma, 2-way automata ).	250
83*.	<i>Decomposition theorems.</i> Product decomposition, modular decomposition (Rüdding normal form in sequential and parallel signal processing).	256

- 84\*. *Small universal programs* 2-dimensional Turing 276  
 machine with 2 states and 4 letters, 2-dimensional  
 Thue-system with 2 rules and 3 letters, PTAPE-complete  
 Loop problem.

PART V CONTEXT-FREE LANGUAGES 297

81. *Normal forms* of Chomsky and Greibach, 297  
*derivation trees.*
82. *Periodicity properties* Loop lemma, Parikh's 305  
 theorem, inductive characterisation through substitution  
 iteration.
83. *Characterisation by machines* Push-down automata, 311  
 closure properties.
84. *Decision problems* Decidability theorem for 316  
 context-free and regular grammars, complexity of the  
 equivalence problem for regular expressions,  
 undecidability theorem for context-free grammars,  
 impossibility of effective minimisation.
- 85\*. *Comparison with the Chomsky hierarchy classes* 327  
 Intersection of regular with bracket languages, L-R  
 derivation restrictions of type-0 grammars, context-  
 dependent languages (space-requirement theorem and the  
 LBA problem).



Book Two 335  
**ELEMENTARY PREDICATE LOGIC**

Chapter D	<b>LOGICAL ANALYSIS OF THE TRUTH CONCEPT</b>	337
<b>PART I. SYNTAX AND SEMANTICS</b>		337
§1.	<i>Formal languages of the first order.</i>	337
§2.	<i>Interpretation of formal languages.</i>	342
§3.	<i>Hilbert calculus.</i>	350
<b>PART II. COMPLETENESS THEOREM</b>		357
§1.	<i>Derivations and deduction theorem for sentence logic</i>	357
§2.	<i>Completeness of propositional logic.</i> (Lindenbaum maximisation process; analytical tables, resolution)	361
§3.	<i>Derivations and deduction theorem of the predicate logic</i>	367
§4.	<i>Completeness of predicate logic.</i>	372
<b>PART III. CONSEQUENCES OF THE COMPLETENESS THEOREM</b>		378
§1.	<i>Weakness of expressibility of PL 1. Theorem of Skolem, compactness theorem, non-characterisability of the concept of infiniteness, non-standard models of number theory.</i>	378
§2*.	<i>Second order predicate logic and type theory.</i> characterisation of finiteness and countability and of $(N; 0, +1)$ in second order; languages of $n$ -th order.	382
§3.	<i>Canonical satisfiability.</i> Skolem normal form, (minimal) Herbrand models, predicate logic resolution, procedural interpretation of Horn formulae, completeness of SLD-resolution.	387
Chapter E.	<b>LOGICAL ANALYSIS OF THE CONCEPT OF PROOF.</b>	400
<b>PART I. GENTZEN'S CALCULUS LK.</b>		401
§1.	<i>The calculus LK.</i>	401
§2.	<i>Equivalence to the Hilbert calculus.</i>	404