
Texts in Computer Science

Series Editors

David Gries, Department of Computer Science, Cornell University, Ithaca, NY,
USA

Orit Hazzan, Faculty of Education in Technology and Science, Technion—Israel
Institute of Technology, Haifa, Israel

More information about this series at <http://www.springer.com/series/3191>

Stefano Crespi Reghizzi •
Luca Breveglieri • Angelo Morzenti

Formal Languages and Compilation

Third Edition

Stefano Crespi Reghizzi
Dipartimento di Elettronica, Informazione e
Bioingegneria
Politecnico di Milano
Milan, Italy

Luca Breveglieri 
Dipartimento di Elettronica, Informazione e
Bioingegneria
Politecnico di Milano
Milan, Italy

Angelo Morzenti
Dipartimento di Elettronica, Informazione e
Bioingegneria
Politecnico di Milano
Milan, Italy

ISSN 1868-0941

ISSN 1868-095X (electronic)

Texts in Computer Science

ISBN 978-3-030-04878-5

ISBN 978-3-030-04879-2 (eBook)

<https://doi.org/10.1007/978-3-030-04879-2>

Library of Congress Control Number: 2018965427

1st & 2nd editions: © Springer-Verlag London 2009, 2013

3rd edition: © Springer Nature Switzerland AG 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This third edition updates, enriches, and revises the 2013 printing [1], without dulling the original structure of the book. The selection of materials and the presentation style reflect many years of teaching compiler courses and of doing research on formal language theory and formal methods, on compiler and language design, and to a lesser extent on natural language processing. In the turmoil of information technology developments, the subject of the book has kept the same fundamental principles since half a century, yet preserving its conceptual importance and practical relevance.

This state of affairs of a topic, which is central to computer science and information processing and is based on established principles, might lead some people to believe that the corresponding textbooks are by now consolidated, much as the classical books on calculus or physics. In reality, this is not the case: there exist fine classical books on the mathematical aspects of language and automata theory, but, for what concerns application to compiling and data representation, the best books are sort of encyclopedias of algorithms, design methods, and practical tricks used in compiler design. Indeed, a compiler is a microcosm, and it features many different aspects ranging from algorithmic wisdom to computer hardware, system software, and network interfaces. As a consequence, the textbooks have grown in size and compete with respect to their coverage of the latest developments in programming languages, processor and network architectures and clever mappings from the former to the latter.

To put things into order, it is better to separate such complex topics into two parts, that we may call *basic* or fundamental, and *advanced* or technological, which to some extent correspond to the two subsystems that make a compiler: the user-language level specific front-end and the machine-level specific back-end. The basic part is the subject of this book; it covers the principles and algorithms to be used for defining the syntax of languages and for implementing simple translators. It does not include: the specialized know-how needed for various classes of programming languages (procedural, functional, object oriented, etc.), the computer architecture related aspects, and the optimization methods used to improve the performance of both the compilation process and the executable code produced.

In other textbooks on compilers, the bias toward practical aspects has reduced the attention to fundamental concepts. This has prevented their authors from taking

advantage of the improvements and simplifications made possible by decades of both extensive use and theoretical polishing, to avoid the irritating variants and repetitions that are found in the historical fundamental papers. Moving from these premises, we decided to present, in a simple minimalist way, the principles and methods currently used in designing syntax-directed applications such as parsing and regular expression matching. Thus, we have substantially improved the standard presentation of parsing algorithms for both regular expressions and grammars, by unifying the concepts and notations used in various approaches, and by extending method coverage with a reduced definitional apparatus. An example that expert readers should appreciate is the unification of top-down and bottom-up parsing algorithms within a new rigorous yet practical framework.

In this way, the reduced effort and space needed to cover classical concepts has made room for new advanced methods typically not present in similar textbooks. First, we have introduced in this edition a new efficient algorithm for string matching using regular expressions, which are increasingly important in such applications as Web browsing and data filtering. Second, our presentation of parsing algorithms focuses on the more expressive extended *BNF* grammars, which are the de facto standard in language reference manuals. Third, we present a parallel parsing algorithm that takes advantage of the many processing units of modern multi-core devices to speed up the syntax analysis of large files. At last, we mention the addition in this edition of the recent formal models of input-driven automata and languages (also known as visibly pushdown model), which suits the mark-up languages (such as *XML* and *JSON*) and has been developed also in other domains to formally verify critical systems.

The presentation is illustrated by many small yet realistic examples, to ease the understanding of the theory and the transfer to application. Theoretical models of automata, transducers, and formal grammars are extensively used, whenever practical motivations exist. Algorithms are described in a pseudo-code to avoid the disturbing details of a programming language, yet they are straightforward to convert into executable procedures, and for some, we have made available their code on the Web.

The book is not restricted to syntax. The study of translations, semantic functions (attribute grammars), and static program analysis by data flow equations enlarges the understanding of the compilation process and covers the essentials of a syntax-directed translator.

A large collection of problems and solutions is available at the authors' course site.

This book should be welcome by those willing to teach or learn the essential concepts of syntax-directed compilation, without needing to rely on software tools and implementations. We believe that learning by doing is not always the best approach and that an overcommitment to practical work may sometimes obscure the conceptual foundations. In the case of formal languages, the elegance and essentiality of the underlying theory allows students to acquire the fundamental paradigms of language structures, to avoid pitfalls such as ambiguity, and to adequately map structure to meaning. In this field, many if not all relevant algorithms

are simple enough to be practiced by paper and pencil. Of course, students should be encouraged to enroll in a hands-on laboratory and to experiment syntax-directed tools (like *flex* and *bison*) on realistic cases.

The authors thank the colleagues Alessandro Barenghi and Marcello Bersani, who have taught compilers to computer engineering students by using this book.

The first author remembers the late Antonio Grasselli, the computer science pioneer who first fascinated him with the subject of this book, combining linguistic, mathematical, and technological aspects.

Milan, Italy
April 2019

Stefano Crespi Reghizzi
Luca Breveglieri
Angelo Morzenti

Reference

1. Crespi Reghizzi S, Breveglieri L, Morzenti A (2013) Formal languages and compilation, 2nd edn. Springer

Contents

1	Introduction	1
1.1	Intended Scope and Readership	1
1.2	Compiler Parts and Corresponding Concepts	2
2	Syntax	5
2.1	Introduction	5
2.1.1	Artificial and Formal Languages	5
2.1.2	Language Types	6
2.1.3	Chapter Outline	7
2.2	Formal Language Theory	8
2.2.1	Alphabet and Language	8
2.2.2	Language Operations	11
2.2.3	Set Operations	13
2.2.4	Star and Cross	14
2.2.5	Language Quotient	17
2.3	Regular Expressions and Languages	17
2.3.1	Definition of Regular Expression	18
2.3.2	Derivation and Language	20
2.3.3	Other Operators	24
2.3.4	Closure Properties of Family <i>REG</i>	25
2.4	Linguistic Abstraction: Abstract and Concrete Lists	26
2.4.1	Lists with Separators and Opening/Closing Marks	27
2.4.2	Language Substitution	28
2.4.3	Hierarchical or Precedence Lists	29
2.5	Context-Free Generative Grammars	31
2.5.1	Limits of Regular Languages	31
2.5.2	Introduction to Context-Free Grammars	32
2.5.3	Conventional Grammar Representations	34
2.5.4	Derivation and Language Generation	37
2.5.5	Erroneous Grammars and Useless Rules	39
2.5.6	Recursion and Language Infinity	41
2.5.7	Syntax Tree and Canonical Derivation	43
2.5.8	Parenthesis Languages	47

2.5.9	Regular Composition of Context-Free Languages	51
2.5.10	Ambiguity in Grammars	52
2.5.11	Catalogue of Ambiguous Forms and Remedies	54
2.5.12	Weak and Structural Equivalence	62
2.5.13	Grammar Transformations and Normal Forms	65
2.6	Grammars of Regular Languages	82
2.6.1	From Regular Expression to Grammar	82
2.6.2	Linear and Unilinear Grammars	84
2.6.3	Linear Language Equations	87
2.7	Comparison of Regular and Context-Free Languages	89
2.7.1	Limits of Context-Free Languages	93
2.7.2	Closure Properties of Families <i>REG</i> and <i>CF</i>	95
2.7.3	Alphabetic Transformations	97
2.7.4	Grammars with Regular Expressions	100
2.8	More General Grammars and Language Families	104
2.8.1	Chomsky Classification of Grammars	105
2.8.2	Further Developments and Final Remarks	109
	References	112
3	Finite Automata as Regular Language Recognizers	115
3.1	Introduction	115
3.2	Recognition Algorithms and Automata	116
3.2.1	A General Automaton	117
3.3	Introduction to Finite Automata	120
3.4	Deterministic Finite Automata	122
3.4.1	Error State and Total Automaton	123
3.4.2	Clean Automaton	124
3.4.3	Minimal Automaton	125
3.4.4	From Automaton to Grammar	129
3.5	Nondeterministic Automata	130
3.5.1	Motivation of Nondeterminism	131
3.5.2	Nondeterministic Recognizers	133
3.5.3	Automata with Spontaneous Moves	135
3.5.4	Correspondence Between Automata and Grammars	136
3.5.5	Ambiguity of Automata	137
3.5.6	Left-Linear Grammars and Automata	138
3.6	From Automaton to Regular Expression: <i>BMC</i> Method	139
3.7	Elimination of Nondeterminism	142
3.7.1	Elimination of Spontaneous Moves	142
3.7.2	Construction of Accessible Subsets	144
3.8	From Regular Expression to Recognizer	147
3.8.1	Thompson Structural Method	148
3.8.2	Berry–Sethi Method	150

3.9	String Matching of Ambiguous Regular Expressions	165
3.9.1	Trees for Ambiguous r.e. and Their Phrases	165
3.9.2	Local Recognizer of Linearized Syntax Trees	172
3.9.3	The Berry–Sethi Parser	172
3.10	Expressions with Complement and Intersection	188
3.10.1	Product of Automata	190
3.11	Summary of the Relations Between Regular Expressions, Grammars and Automata	195
	References	196
4	Pushdown Automata and Parsing	199
4.1	Introduction	199
4.2	Pushdown Automaton	200
4.2.1	From Grammar to Pushdown Automaton	201
4.2.2	Definition of Pushdown Automaton	205
4.2.3	One Family for Context-Free Languages and Pushdown Automata	209
4.2.4	Intersection of Regular and Context-Free Languages	213
4.3	Deterministic Pushdown Automata and Languages	215
4.3.1	Closure Properties of Deterministic Languages	216
4.3.2	Nondeterministic Languages	218
4.3.3	Determinism and Language Unambiguity	220
4.3.4	Subclasses of Deterministic Pushdown Automata and Languages	221
4.4	Syntax Analysis: Top-Down and Bottom-Up Constructions	230
4.5	Grammar as Network of Finite Automata	233
4.5.1	Syntax Charts	237
4.5.2	Derivation for Machine Nets	239
4.5.3	Initial and Look-Ahead Characters	241
4.6	Bottom-Up Deterministic Analysis	244
4.6.1	From Finite Recognizers to Bottom-Up Parser	245
4.6.2	Construction of <i>ELR</i> (1) Parsers	250
4.6.3	<i>ELR</i> (1) Condition	253
4.6.4	Simplifications for <i>BNF</i> Grammars	267
4.6.5	Parser Implementation Using a Vector Stack	271
4.6.6	Lengthening the Look-Ahead	275
4.7	Deterministic Top-Down Parsing	282
4.7.1	<i>ELL</i> (1) Condition	286
4.7.2	Step-by-Step Derivation of <i>ELL</i> (1) Parsers	288
4.7.3	Direct Construction of Top-Down Predictive Parsers	305
4.7.4	A Graphical Method for Computing Guide Sets	314
4.7.5	Increasing Look-Ahead in Top-Down Parsers	322

4.8	Operator-Precedence Grammars and Parallel Parsing	325
4.8.1	Floyd Operator-Precedence Grammars and Parsers	326
4.8.2	Sequential Operator-Precedence Parser	330
4.8.3	Comparisons and Closure Properties	333
4.8.4	Parallel Parsing Algorithm	339
4.9	Deterministic Language Families: A Comparison	346
4.10	Discussion of Parsing Methods	350
4.11	A General Parsing Algorithm	352
4.11.1	Introductory Presentation	353
4.11.2	Earley Algorithm	357
4.11.3	Syntax Tree Construction	366
4.12	Managing Syntactic Errors and Changes	373
4.12.1	Errors	373
4.12.2	Incremental Parsing	379
	References	379
5	Translation Semantics and Static Analysis	383
5.1	Introduction	383
5.2	Translation Relation and Function	386
5.3	Transliteration	388
5.4	Purely Syntactic Translation	389
5.4.1	Infix and Polish Notation	391
5.4.2	Ambiguity of Source Grammar and Translation	395
5.4.3	Translation Grammar and Pushdown Transducer	397
5.4.4	Syntax Analysis with Online Translation	402
5.4.5	Top-Down Deterministic Translation by Recursive Procedures	403
5.4.6	Bottom-Up Deterministic Translation	406
5.5	Regular Translation	414
5.5.1	Two-Input Automaton	416
5.5.2	Translation Function and Finite Transducer	421
5.5.3	Closure Properties of Translation	426
5.6	Semantic Translation	427
5.6.1	Attribute Grammars	429
5.6.2	Left and Right Attributes	431
5.6.3	Definition of Attribute Grammar	435
5.6.4	Dependence Graph and Attribute Evaluation	437
5.6.5	One-Sweep Semantic Evaluation	442
5.6.6	Other Evaluation Methods	446
5.6.7	Combined Syntax and Semantic Analysis	447
5.6.8	Typical Applications of Attribute Grammar	457

5.7	Static Program Analysis	467
5.7.1	A Program as an Automaton	468
5.7.2	Liveness Intervals of a Variable	471
5.7.3	Reaching Definition	477
	References	486
Index		487