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

Springer

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